Novel Learning and Exploration-Exploitation Methods for Effective Recommender Systems

Romain Warlop

Université des Sciences et des Technologies de Lille

Thèse préparée et soutenue publiquement par
Romain WARLOP

Pour obtenir le grade de
Docteur en Informatique et Applications

Novel Learning and Exploration-Exploitation Methods for Effective Recommender Systems

Sous la direction de: Jérémie MARY et Alessandro LAZARIC

Membres du jury:

Florence	**D'ALCHE-BUC**	Telecom Paristech	*(Examinateur)*
Rémi	**GILLERON**	Université de Lille	*(Examinateur)*
Alessandro	**LAZARIC**	Facebook / Inria	*(Co-Directeur)*
Jérémie	**MARY**	CRITEO / Université de Lille	*(Directeur)*
Arnaud	**MASSONNIE**	FIFTY-FIVE	*(Invité)*
Liva	**RALAIVOLA**	Université de Marseille/IUF	*(Rapporteur)*
Michèle	**SEBAG**	CNRS Saclay	*(Rapporteur)*
Jean-François	**WASSONG**	FIFTY-FIVE	*(Invité)*

Ecole Doctorale SPI
Université Lille Nord de France
Bât. P3 (bureau 304-306)
F 59655 Villeneuve d'Ascq cedex
Tél : (33)3-20-43-65-19

October 19th, 2018

Remerciements

Le travail réalisé au cours de ces trois années de thèse n'aurait jamais pu avoir lieu sans la présence et l'accompagnement de nombreuses personnes.

Mes remerciements vont tout d'abord à mes deux maîtres de thèse, Alessandro Lazaric et Jérémie Mary, d'accepté de me suivre dans cette aventure de thèse CIFRE quelque peu originale. Nos discussions m'ont énormément apporté. J'ai eu l'impression de redécouvrir le machine learning tous les vendredis.

Je remercie grandement Michèle Sebag et Liva Ralaivola d'avoir accepté d'être mes rapporteurs, surtout que la période n'était pas forcément la meilleure pour relire une thèse. Je remercie aussi Florence D'Alche-Buc et Rémi Gilleron d'avoir accepté d'être membres du jury.

Je remercie également toutes les personnes de l'équipe SequeL qui m'ont accueilli dans leur bureau, notamment Michal Valko, toujours à l'écoute de ma recherche et prêt à aider de toutes les manières possibles.

Cette thèse CIFRE n'aurait évidemment pas pu exister sans la présence de fifty-five. Un grand merci à Arnaud Massonnie pour ce déjeuner où tu m'as aiguillé sur le droit chemin, pour ces déjeuners où tu as écouté avec attention toutes mes propositions les plus improbables. Chacune de ces discussions me redonne toujours de l'énergie, de l'envie et de la confiance. Merci à Jean-François Wassong, toujours présent pour écouter également mes idées aléatoires et soucieux de l'équilibre entre mon travail de recherche et mon travail industriel.

Merci à mes collègues de fifty-five, notamment ceux de mon équipe. Merci Martin et Paul de m'avoir fait grandir en data science. Merci Bettina et Bridget pour tous les projets qu'on a fait ensemble et toutes ces pauses qui m'ont bien aidé pendant ces années... Sans vous quatre ces années auraient été bien tristes.

Merci à mes anciens colocs++, Chris, Keurcien, Matthieu, Maxime, Pierre (t'étais pas à la coloc mais je te compte dedans quand même), Thibaud, et surtout William bien sûr. Merci pour nos vacances, votre soutien à tout moment pour tout problème, pour Fifa, pas pour LoL, pour la coloc et pour tout ce qu'on va vivre ensemble dans le futur. Merci également à Anaïs de parler plus vite que moi et à Aude pour m'avoir apporté des Monster Munch.

Henri, la prépa c'est censé être dur, les colles aussi, mais passer deux ans à partager mes colles avec toi, c'était le pire. Heureusement j'ai fait 5/2 et pas toi, j'ai pu profiter un peu de la prépa. Tu te souviens de rien alors je te donne quelques-uns de mes souvenirs: merci pour le Michelson, pour notre première colle avec Charmont, de m'avoir insulté après t'avoir dit "à tes souhaits" en colle avec Dumas.

Merci à ma famille, papa, maman, Lucie, Vincent, qui m'a soutenu durant toute ma scolarité, merci de m'avoir toujours poussé. Merci maman pour tout ce que tu as fait de la maternelle à l'ENS en passant par la prépa où tu as passé beaucoup de ton temps à faire en sorte que j'en aie.

Merci à Mimi, Zizie et bien évidemment Melwenn. Merci d'avoir laissé ta lumière allumée, merci de m'avoir écouté soir après soir parler de ma thèse, je n'en parlerai plus. Merci d'avoir relu ma thèse. Merci d'aimer les chats. Merci d'avoir de la caisse.

Merci à tous.

Résumé court en français. Cette thèse, réalisée en entreprise en tant que thèse CIFRE dans l'entreprise *fifty-five*, étudie les algorithmes des systèmes de recommandation. Nous avons proposé trois nouveaux algorithmes améliorant l'état de l'art que ce soit en termes de performance ou de prise en compte des contraintes industrielles. Pour cela nous avons proposé un premier algorithme basé sur la factorisation de tenseur, généralisation de la factorisation de matrice couramment appliquée en filtrage collaboratif. Cette extension permet de prendre en compte simultanément différents types d'interaction entre les utilisateurs et les produits et dans des contextes différents. L'algorithme proposé est également hautement parallélisable ce qui le rend facilement utilisable sur des données réelles très volumineuses. Nous avons ensuite proposé un nouvel algorithme permettant d'améliorer l'état de l'art des solutions de complétion de paniers. L'objectif des algorithmes de complétion de paniers est de proposer à l'utilisateur un nouveau produit à ajouter au panier qu'il/elle est en train d'acheter permettant ainsi d'augmenter la valeur d'un utilisateur. Pour cela nous nous sommes appuyés sur les processus ponctuels déterminantal (DPP), c'est à dire une mesure de probabilité dont la probabilité d'observer un ensemble est proportionnel au déterminant d'un noyau. Nous avons généralisé l'approche de la complétion de paniers par DPP en utilisant une approche tensorielle couplée à une régression logistique. Enfin nous avons proposé un algorithme d'apprentissage par renforcement permettant d'alterner entre différents algorithmes de recommendation. En effet, utiliser toujours le même algorithme peut avoir tendance à ennuyer l'utilisateur pendant un certain temps, ou à l'inverse lui donner de plus en plus confiance en l'algorithme. Ainsi la performance d'un algorithme donné n'est pas stationnaire et dépend de quand et à quelle fréquence celui-ci a été utilisé. Nous avons alors modélisé la performance future d'un algorithme par une régression linéaire définie par un polynôme en une fonction de récence, c'est à dire une fonction qui mesure la fréquence d'utilisation d'un algorithme dans un historique récent. Notre algorithme d'apprentissage par renforcement apprend alors en temps réel à alterner entre divers algorithmes de recommendations dans le but de maximiser les performances sur le long terme, c'est à dire de continuer d'intéresser l'utilisateur le plus longtemps possible. Cet algorithme peut être vu comme un algorithme de recommendation hybride.

Cette thèse ayant été réalisée en entreprise, nous avons toujours recherché à respecter les contraintes industrielles lors du développement de nouvelles solutions. Ainsi chaque chapitre présentant un nouvel algorithme contiendra une section dans laquelle nous présenterons comment la solution a été utilisée ou pourrait être utilisée en pratique.

Short English abstract. This thesis, written in a company as a CIFRE thesis in the company *fifty-five*, studies recommender systems algorithms. We propose three new algorithms that improved over state-of-the-art solutions in terms of performance or matching industrial constraints. To that end, we proposed a first algorithm based on tensor factorization, a generalization of matrix factorization, commonly used on collaborative filtering. This extension allows to take into account simultaneously several types of feedbacks as well as different contexts. The proposed algorithm is also highly parallelisable thus suitable for real life large datasets. We then proposed a new algorithm that improves basket completion state-of-the-art algorithms. The goal of basket completion algorithms is to recommend a new product to a given user based on the products she is about to purchase in order to increase the user value. To that end we leverage Determinantal Point Processes (DPP), i.e., probability measure where the probability to observe a given set is proportional to the determinant of a kernel matrix. We generalized DPP approaches for basket completion using a tensorial point of view coupled with a logistic regression. Finally, we proposed a reinforcement learning algorithm that allows to alternate between several recommender systems algorithms. Indeed, using always the same algorithm may either bore the user for a while or reinforce her trust in the system. Thus, the algorithm performance is not stationary and depends on when and how much the algorithm has been used in the past. We then model the future performance of an algorithm according to linear function which is a polynomial in a recency function, that is a function that measures the frequency of use of an algorithm in a recent history. Our reinforcement learning algorithm learns in real time how to alternate between several recommender system algorithms in order to maximize long term performances, that is in order to keep the user interested in the system as long as possible. This algorithm can be seen as an hybrid recommender system.

This thesis having been written in a company, we always looked for considering industrial contraints when developing new algorithms. Thus, each chapter that introduces a new algorithm will contain a section in which we present how the solution has been used or could be used in practice.

Contents

List of Figures

List of Tables

fifty-five

This PhD was written inside the company FIFTY-FIVE as part of a CIFRE (Conventions Industrielles de Formation par la REcherche) project.

Part of You & Mr Jones, the world's first brandtech group, FIFTY-FIVE is a data company helping brands to collect, analyse and activate their data across paid, earned and owned channels to increase their marketing ROI and improve customer acquisition and retention. Headquartered in Paris with offices in London, Hong Kong, New York and Shanghai, the data company was named by Deloitte as one of the fastest-growing tech firms in Europe, thanks to its unique technology approach combining talent with software and service expertise.

FIFTY-FIVE data expertise is leveraged by data science in order to solve marketing issues in order to optimize user value thanks to product recommendation, website personnalization, churn detection, marketing user clustering or website pain points detection.

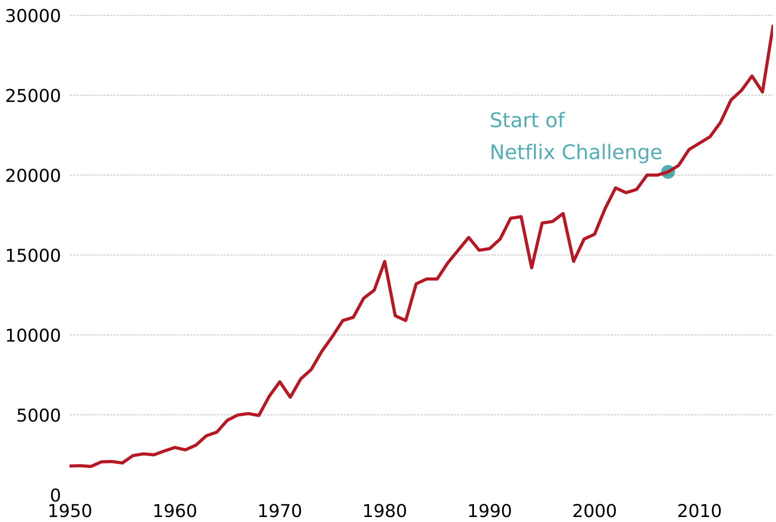
Introduction

1.1 Overview of Recommender Systems

People who liked this product also liked ... - According to your recent browsing history you may like ... - Because you watch this movie you may also enjoy ... With the increasing amount of choices, especially on the internet, recommender systems play an important role in helping users to find relevant items according to their taste and such advertising are spreading everywhere. Indeed having a tremendous amount of choices creates a positive feeling of freedom but also overwhelms users and leads them to make poor decisions [Ricci et al., 2010]. To overcome this problem, recommender systems have emerged at the intersection of computer science and the machine learning community, in order to guide the user into her search. Many companies use recommender systems with success everyday like e-commerce websites (Amazon, Wallmart), News websites (e.g. Yahoo), Subscription Video On Demand services (e.g. Netflix, Amazon, HBO Go, Hulu), crowd-sourcing reviews providers (e.g. Yelp, TripAdivsor), Massive Open Online Courses (e.g. Coursera, edX), music streaming services (e.g. Spotify, Deezer, SoundCloud), retargetings (e.g. Criteo) in order to provide personalized content to each user and improve user experience.

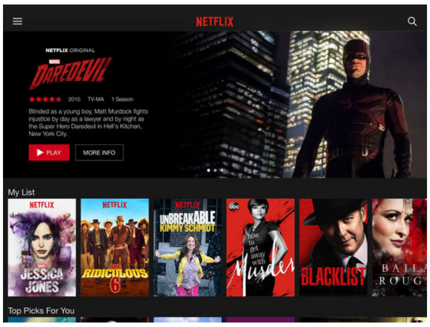
The research field of recommender systems emerged in the mid-1990s, and has kept increasing ever since, with thousands of research papers on the topic (see Figure 1.1). Conferences are now dedicated to this field like the ACM Recommender System (RecSys) since 2007, and international machine learning and artificial intelligence conferences like ICML, AAAI, KDD, NIPS, ..., include sessions dedicated to recommender systems. However recommendation was not invented with machine learning and the Internet. Everyday we get recommendations from people around us who share their experience that we trust more or less given our personal similarity with the person and the information they provide on the experience. Before watching a movie we read newspapers critics to make our decision, and likewise we are more willing to trust some journalist or newspaper than others. That is what

Figure 1.1: Number of match by year of the search *Recommender+system* on Google Scholar



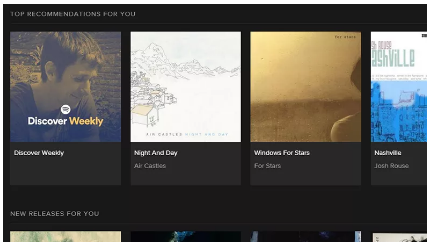
recommender systems are trying to mimic by finding similarities between thousands or millions of users in order to collaboratively infer our tastes on items we do not know of but some similar - yet to define - users consumed and filter out the possibilities. The underlying simple idea being: if we agreed in the past, we are likely to agree in the future.

In order to perform effective recommendations, machine learning algorithms leverage knowledge from historical data to infer rules, correlations, descriptors, preferences and use those to pick the right item(s) to present to each user, where an "item" is the generic term to denote what the system is trying to recommend to users, such as books, songs, news, movies, clothes, restaurants, ... Historical data can be very heterogeneous, such as specific information on the user (i.e., gender, age, occupation, zip), item characteristics (i.e., genre, year, actors, color, texture, price, discount), explicit feedback from users on items (i.e., rating, like/dislike) that indicate their feeling or implicit feedback gathered through users interactions with items (i.e., view, click, add to cart, purchase, zoom) that can be interpreted as a sign of interest. By interacting with the system, the user keeps giving new information about its preferences that is used to update the system and provide better future recommendations. Leveraging this information to provide a personalized experience to users is the challenge of recommender systems. This dissertation aims at improving state-of-the-art solutions in adequation with industrial constraints.



(a) Netflix

(b) Amazon



(c) Spotify

Figure 1.2: Some real world recommender systems

1.2 Limitations and open questions

On October 2, 2006 the Netflix Challenge officially started. IT rewarded anyone who improved the accuracy (in terms of RMSE) of their current algorithm, called Cinematch, by 10% with a million dollar price. During three years many teams and people gathered and competed to propose recommender system solutions. This led to many valuable lessons but also showed the limitations of current methods and introduced many open questions.

In this challenge, matrix factorization techniques proved their effectiveness to address the recommendation problem and led to many tuning in the following years. Among others, taking into account the date in multiple ways also showed interesting improvements underlying non-stationarity in recommender systems. Finally blending many different algorithms, i.e ensemble methods, proved their effectiveness and was the key to win the competition. Indeed, the final winning submission is actually made of more than 500 algorithms where each algorithm has good performance in some specific regions of the possibilites. Merging all the predictions according to extra knowledge, like the number of feedback the user had previously provided, the number of feedback the movie had received, the date, etc ..., gave the extra-performance boost to beat Cinematch by more than 10%.

However several critics have arised during and after this challenge. First, the ensemble method proposed by the winning team is actually unusable in practice - and is not used by Netflix - because it requires far too much computational power to be used in the real world. Second the choice of the evualation metric, the root mean squared error, was critized because it does not really reflect how the system will be used in practice. Following work then focused on different optimization solutions, like learning to rank instead of learning to predict an affinity, and different evaluation metrics that we will detail in the next chapter.

Besides those critics, state-of-the-art recommender system algorithms suffer from other limits that nurtured reflection for this thesis. The **first limit** that penalize recommender systems is the poverty of user historical data, called **cold-start** problem. Indeed in web application, user identication is performed using cookie tagging (a small code placed in the user browser). Those cookies have an expiration date of 13 months by law, are not consistent through different devices - that is a user using two devices will be seen as two different users - and they can be manually deleted. Since users preferences are learned using past feedback on items, users in cold-start situations will receive poor recommandation. This problem is less true for applications were people have to login in order to access the website and have frequent interactions like Netflix or Facebook. To overcome this issue we extend state-of-the-art algorithm 3.6.1 to take into account **multiple types of interactions** within the same framework and thus limit the poverty of available feedback per user. Indeed in practice users can interact with items in a various number of ways and thus limiting the analysis to only one type of feedback lessens the possibilities.

A **second limit** is the computational complexity of basket completion algorithm. State-of-the-art solutions are mostly based on explicitly computing co-occurences that are untractable in practice. Some solutions have been developed in the recent years based on Determinantal Point Process and on Recurrent Neural Network for the more general problem of session based recommender system (see 4.5.1). In this dissertation, we propose a new algorithm that adresses the basket completion issue and opens the door to more generalization of Determinantal Point Processes.

A **third limit** comes from online model evaluation. Usually people perform tests, that is different users will be exposed to a different model, but always the same one for a given user, and success rate is monitored through time to output the best model. Comparing several solutions in order to find the best one is a very active domain since exploration optimization among a (fix) set of solutions is the core interest of the bandit community (section 2.1.3). However the question of the non stationary behaviour of the action (here the recommender system that is being tested) and the impact of past pulled (past recommender systems choices for a user) on next pull performance had received little attention in recommender system and will be studied in this dissertation.

Partially adressed by this dissertation, the **filter bubble** problem is a main limit of recommender system. This designed the fact that recommender systems lock user in their past tastes and prevent them from discovering new areas and diversifing their viewpoints. To adress the question, machine learning researcher focus on the trade off between **exploitation** (taking the best action) and **exploration** (gathering new information) without degrading too much short term reward and hoping to maximize future long term ones.

1.3 Overview of the dissertation

In the rest of the dissertation we first provide more machine learning background on recommender systems before presenting various works that address previoulsy introduced state-of-the-art recommender systems limitations.

- chapter 2 provides more machine learning related background on recommender systems. After introducing the different purposes and categories of recommender systems, we then introduce the classic measurement protocol for performance evaluation that will be used in the experiments.

- chapter 3: this chapter proposes an extension of matrix factorization based algorithm to tensor factorization in order to deal with multiple types of feedback at the same time and alleviate the cold-start problem. We proposed two algorithms for Tucker and parafac tensor factorization, respectively called tucker and parafac Hoals that are highly parallelizable and appropriately deal with missing data, two problems that were not correctly tackled by tensor factorization state-of-the-art. Performance evaluation on open data and fifty-five propriaritary data highlight the effectiveness of the proposed methods, that also have the major advantage of being easily implemented using available matrix factorization libraries like spark recommendation module.

- chapter 4: this chapter introduces an extension of Determinantal Point Process algorithms, called Multi-Task dpp, for basket completion with two complementary updates that allows more accurate basket completion that is also scalable to large item catalogs. The first one consists in using a logistic function upon the Determinantal Point Process to better model the supervision task. The second one is a low rank tensor approach for multi task prediction. Although we focused on basket completion problem, where each task is a candidate item to add to someone basket, this algorithm can be tested for many different applications and can be the starting point to other generalizations of Determinantal Point Process for machine learning.

- chapter 5: always using the same strategy to personalize user experience is likely to cause users boredom or more generally non-stationary performances because of the evolution of user's preferences. A better idea is then to alternate between several personalization strategies. For instance, one could use an algorithm that focuses on serendipity, or on recommending very similar items. Changing the number of recommended items can also fight users' boredom. Which variation to test and when to change is however an open question. This chapter proposes an algorithm, called LinUCRL to effectively learn the right way to alternate between several available algorithms in order to maximize user interest.

All those contributions are ultimately used together to fully personalize the user interaction with the brand at different level of its engagement. At first, when the user is completly unknown from the brand, fifty-five does not hold any personal feedback from this user to provide good personalisation. The user is then engaged using different media strategies like retargeting. Once the user reaches the website, a first customization is performed using popularity based algorithms until the user reaches the basket page where basket completion algorithms, like Logistic dpp, are used to increase the user value. Now that the user is known to the brand, fifty-five can use other recommender system algorithms, like the ones of Hoals, first on e-mailing or mobile application notification, to give the user the incentive to come back on the website, where those algorithms are again used on the landing page. The journey now enters to a loop when the user reaches the basket page and basket completion algorithms are used again. At each personalization stage, LinUCRL algorithm is used to fight user boredom.

1.4 From an algorithm to a recommendation system

In a general sense, a good algorithm is not enough to have a good recommender system. Indeed feedbacks must be collected to feed the algorithm and recommendations must be displayed to the users. Thus the place of the user, the choice of the interface, the way data is collected, treated, used is essential. Here we discuss those questions that each brand asks whenever recommender systems come up into the discussion.

User interface and feedback collection

First recommender systems applications like BellCore's video recommender system used to give recommendation through e-mail: users had to e-mail their ratings on movies and would receive back an e-mail with recommendations ([Hill et al., 1995]). While quite obsolete this technique is still used by various e-commerce websites or in the food industry, but it asks too much effort for the user leading to low response rates thus limiting the interest of the method.

More modern solutions put more effort into the design of the interface both to display recommendations and collect feedbacks. For instance, Youtube and Netflix use recommender systems to fully personalize the user's content and use a similar display technique: the items are ordered by row where each row corresponds to items that are related according to some notion of similiarity, such as the same genre, a common actor, related to an other movie, ... This display technique has the advantage of simplifying the navigation so that a user can skip categories that she is not interested in at the moment. Furthermore this particular display also provides new types of feedbacks, since scrolling in a row also shows users' interest in a specific theme. Youtube and Netflix present the advantage of having very engaged signed-in users with lots of navigation, thus an easy way to know the characteristics of their user base. Other companies do not have this advantage and cannot achieve such a deep level of personalization. Banners are then used at different moments of the navigation, like on the home page, on the product page, or on the basket confirmation page, to provide personalized contents with few items along with a catch phrase like "Users who liked this also liked ...". The navigation interaction collected from those interfaces provide implicit user interest with items, called implicit feedbacks. But gathering explicit feedbacks that is an explicit information from the user on how much she likes the item, is always valuable for the system. Some websites like Jester ([Goldberg et al., 2001]) provided a continuous bar where the level informs the users' affinity with the joke. However in most applications binary feedbacks, like thumbs up and down, are used. Indeed, even Netflix let down its world famous 5 star rating system in 2017 mainly for clarity reasons. In fact rating a movie 4 stars does not mean the same for everybody. From a machine learning point of view this problem is easy to solve since it only requires scaling each user ratings around 0 and now 0 means average for everybody. The second problem comes from ambiguity with other websites like IMDB or Amazon for instance. Indeed when Netflix presents a new movie to some users with 4 stars, for them it says "I think that you will rate this movie 4 stars" whereas for others it meant "The rest of world gave it 4 stars". That is the main reason why Netflix adopted the thumbs up/down rating system and provides recommendation with an affinity score as a percentage. This change led to lots of debate on online social networks with people threatening to leave Netflix because they wanted to give 2 stars to some movie. A survey from the *exstreamist* reported that 71% of the Netflix subscribers dislike the new rating system. The main critic is that people want to have more flexibility in the rating system instead of just yes or no. This event highlights the importance of the rating system and the difficulty to find a good one. The thumbs up/down system is also used on Youtube and Spotify for instance, but obtaining a good explicit feedback it still an open question... However for some applications like TV shows or music, explicit feedbacks are not the most valuable information. Indeed, the completion rate of the TV show or listening frequency to a singer are much more informative than a rating or a thumb up. Someone may rate some TV shows 2 by shame but actually love to watch it. Implicit feedbacks are more reliable in that case.

Two other questions always arise when displaying recommendations are: how many items to show and should only new items be displayed? In practice there is no general truth and testing on several options is required. For instance, some users will be exposed to banners/e-mails with 4 items, others with 6. In some displays there will be 1 or 2 items that the users already interacted with, in some others none. Previous tests showed that it is always a good idea to show at least 1 item that the user knows in order to catch the eye and give her confidence that the recommendation can suit her.

Privacy and security issues

As we have previously seen, feedbacks are the cornerstone of recommender systems. The more feedbacks and detailed information the system has about users and items, the more accurate it will be. But all this information raises privacy concerns. The question on how to keep privacy in recommender systems and trade-off between personalization and data privacy is attracting the interest of the machine learning community and in 2017, the RecSys conference dedicated a tutorial on the question. [Ramakrishnan et al., 2001], [McSherry and Mironov, 2009a], [Lam et al., 2006], [Koene et al., 2015] discuss several issues regarding privacy and security.

A first privacy issue can come from hacking attacks that could leak highly sensitive information. Collecting less information about the user without degrading the performance too much was proposed to prevent this risk from occuring. The concept of VOI metric (Value Of Information) was introduced in order to tune a system to optimize data collection on precise items and stop collecting once the system has enough information on the user to make good personalized recommendations. Other kinds of attacks are also a concern in recommender systems. For instance shilling attacks, where someone gives high positive feedbacks to its own products that are sold on a platform in order to increase its chance to be recommended to others and high negative feedbacks to competitors. Creating lots of fake accounts to provide feedbacks is also a well known attack called Sybil attack. Such attacks can bias the model toward suboptimal recommendations. In 2002 Amazon was the victim of a similar attack when a few motivated users intentionally viewed the page for a spiritual guide by Christian televangelist Pat Robertson and "The Ultimate Guide to Anal Sex for Men" one after the other in order to push recommendation of the latter on the page of the former. Being able to identify such attacks are of main concern in recommender systems applications.

A second issue is identification or re-identification that enables to uniquely identify someone based on her feedbacks without any explicit Personally Identifiable Information (PII). [Lam et al., 2006] also mention the risk of quasi-identifiers, i.e., information that can almost uniquely identify a person. For instance, the triplet (zipcode, birth data, gender) is a quasi-identifier. Massachusetts governor William Weld former governor was re-identify with this technique between a voter registration and a supposedly anonymous medical record dataset given to researchers [Sweeney, 2002]. [Narayanan and Shmatikov, 2006] proposed two algorithms to identify records in a public database using an auxiliary partial knowledge on some records and proved successful identifications on the Netflix dataset. With this issue also comes the problem of inference. For instance [Kosinski et al., 2013] shows that highly sensitive information like sexual orientation, political views and age among others can be retrieved using Facebook Likes. In order to cope with those problems several approaches uses randomization and cryptographic techniques ([Polat and Du, 2005, McSherry and Mironov, 2009a, Erkin et al., 2011, Dwork, 2008, Berkovsky et al., 2007, Shokri et al., 2009, Aïmeur et al., 2008, McSherry and Mironov, 2009b]) to trade off between privacy, accuracy, and efficiency. Another solution is user-centric solutions. For instance [Canny, 2002] relied on a model where item latent factors are first locally downloaded by the user, then her own machine performs relevant computation according to its locally stored information in order to find the items that should be recommended to her. After interacting with the system, updates are done using encryption techniques. However such solutions can hardly be used in practice because of very large catalog size which prevents computation from being done on the user's machine and because of product availabilities that evolve almost in real time, and thus that require frequent calls to an external server to get such information.

Outside machine learning research, some commissions exist to try to guarantee user's privacy and rights or at least regulate how data is gathered and exchanged. In France this role is led by the CNIL (Commission Nationale de l'Informatique et des Libertés or National Commission of Informatics and Freedom) and has six main missions: inform, regulate, protect, control, sanction and anticipate. The Federal Trade Commision is the American counterpart but is not focused solely on informatics since its main mission is the promotion of the consumer protection in general. More recently, European Union developed a General Data Protection Regulation (GDPR), released in 2018, to define a set of text laws focusing on personal data processing in order to provide data protection and privacy for all individuals inside the European Union (not only European Citizens). We can also cite e-Privacy, a European text regulation released in 2002, specific to online data and more precisely cookie management.

Filter bubble issues

Filter bubble is a term coined by the internet activist Eli Pariser in 2010 [Pariser, 2011]. In his book he argues that with recommender systems like Google, Facebook or Yahoo, users experience a very personalized content that is different from everyone else. This creates a unique filter bubble for everyone that is invisible and hard to escape from. Such critics have also been raised for the US 2016 election of Donald Trump ([Baer, 2016], [Jackson, 2017], [Hern, 2017]). Those articles claim that users with similar opinions as a candidate will only see news in favour of this candidate and will therefore reinforce their opinion. Besides if their candidate loses, they will not see the defeat arrive since they only saw articles communicating positively about this candidate. Unfortunately little research has been dedicated to investigate this question. Indeed today's research is mainly driven by available data where filter bubble is difficult to test. To the best of our knowledge, the only study on the filter bubble's impact on diversity has been made by [Nguyen et al., 2014]. In this article, authors use the movieLens dataset where one can have access to which movies have been recommended to which users. They then defined a methodology to measure the impact of the recommendation on the user content diversity and satisfaction comparing group of users that used the recommendation with a similar strength. Surprisingly, the narrow effect has been observed not only for users that used the recommender systems but also for users who did not and the diversity reduction was lower for user that used the recommender system than for others. However the movieLens website is quite particular since it is not a consumption website but only a rating platform where it is impossible to know if a user rated a movie because she just watched it or because she thinks of rating it now but watched it a while ago. Integrating filter bubble consideration into every day recommender systems in production would be an interesting step. Coping with filter bubble remains an issue that recommender systems should address. This can be done by introducing diversity in the proposed items at the recommendation time. For instance a solution is to compute a trade off between the item relevance and its similarity with previously displayed items. Similarly, selecting globally diverse sets of items is studied using mathematicals tools borrowed from statistical physics used to model repulsion between particles. Finally, improvement can also be done on the transparency of the user interface that could indicate the level of similarity of an item with a previously consumed one and highlight dissimilar ones, but this could also come with a performance cost that the company may not be willing to take.

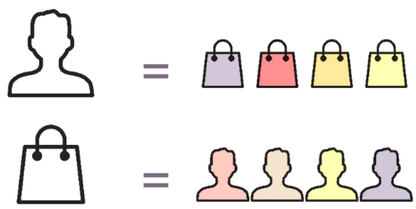
After working at fifty-five, first as an intern then full time, I became quickly interested in the user personnalization problem. However facing real world application constraints, those limitations prevent me from efficiently applying state-of-the-art solutions. Extending state-of-the-art solution to match industrial constraints and objectives was then a main motivation and focus of this work. Consequently the starting point of all reflection was fifty-five available data, the data distribution and the production constraints. In following contribution we will then highlight in a decicated section the industrial learnings that this work brought.

Recommender Systems

2.1 Machine Learning in Recommender Systems

Broadly speaking, the objective of recommender systems algorithms is to present relevant items to users where relevance can have two different meanings. The first one is measured in terms of **affinity**, thus the goal is to show item(s) that users may like. The second one is measured in terms of **coherence** with other items that users will consume together. The main difference from a data perspective is that affinity will be captured based on all the interactions between users and items no matter when they occured, while for coherence the algorithm will focus on co-occurence for instance in the same session for web application. Both objectives require past data in order to learn relevance of items in each context in order to present those items to users. Thus future data will directly depend on the recommender system past choices which can bias future learning. To avoid such problem, recommender systems must balance between exploration, that is testing different possibilities to gather new information, and exploitation, that is acting according to previously

(a) Content Based model - users and items are described with contextual information.



(b) Collaborative Filtering model - users are described based on their past interactions with items, items are described based on the past users that interacted with them.

Figure 2.1: Illustation of content-based and collaborative filtering models.

gathered data. Bandit learning provides a framework in order to effectively balance exploration and exploitation.

2.1.1 On predicting user-item affinity

In order to predict future user-item affinities, algorithms rely on any available information on users and items: this could be any kind of contextual information like age, gender, genre, date, or previous user-item interaction. Once the model is learned, the system outputs the affinity between one user and either all possible items or only items unknown to the user and picks the best one(s), that is the one(s) with the highest affinity. Indeed, for some applications, like movie recommendation, it does not make sense to recommend a movie that the user has already seen and rated in the past, thus the recommendation will focus on selecting new items. In other applications, like music recommendation or restaurant recommendation, the system may recommend a previously known item if it is predicted to be the most suitable at the current moment.

There are three families of recommender systems that depend on how the algorithm learns affinities between users and items:

- content-based model (see figure 2.1a): user and item are characterized by contextual knowledge. For instance a user is defined by its gender, age, town of residence, occupation, ..., and an item by its color, shape, recency, ...,
- collaborative filtering (see figure 2.1b): user (respectively item) is characterized by the items they interacted with (respectively users that interacted with it),
- hybrid model: mix several models together.

Content-based

Content-based models ([Pazzani and Billsus, 2007]) are simple models to perform recommendation. Any supervised learning techniques can be used in order to perform the recommendation. In supervised learning one assumes that there is a functional relationship between the output Y (or the endogenous variable) and the input X (or the (explicative) variables, the exogenous variables), this function being noted f^*. The goal is to find the best approximation $\hat{f} \in \mathcal{F}$, with $\mathcal{F}$ a set of possible functions, of the function f^* based on past observations, i.e the training set, $(X_{train}, Y_{train}) \in \mathbb{R}^{n \times p} \times \mathbb{R}^n$, with n the number of samples and p the number of variables, that is one must solve:

$$\hat{f} = \arg\min_{f \in \mathcal{F}} d\left(Y_{train}, f(X_{train})\right) \tag{2.1}$$

where d is a distance function.

For recommender systems, the output variable is the feedback provided by a user on an item and the input variables are the descriptive vector of the user and

x							y
male	female	age	action	thriller	drama	romantic	output
1	0	32	0	1	1	0	3

Table 2.1: Content-based input training example

the item. For instance if the contextual variables available for a user are her gender and age and for the item (let us consider a movie) the genre among action, thriller, drama and romantic. The length of the input vector is then 7 where the first 3 values encode the user characteristics and the last 4 values encode the item features. For example, if a 32-year-old male user rated a thriller-action movie 3 stars, we would get the (x, y) training point of table 2.1.

These models present the advantage of being very simple, easy to interpret and and easy to implement by leveraging available supervised learning libraries and techniques. Furthermore, for web applications, the user's information can easily be stored in a cookie and has the advantage of being fix in time (or at least do not change very quickly like age) thus real-time prediction is made easier. In practice content-based methods are used in specific cases when many users have provided little feedback on items, when rich contextual information is available for users and items, or when computing individual predictions is not possible because of time-response constraints. In many e-commerce applications, there is not that much contextual information available on products. Price, discount, color, brand and few product categories are available, but this is still too little to learn a model using just those information. To overcome this problem, deep learning is frequently used to compute latent descriptors of the product image and/or the product text, which results in a good item descriptor ([Grbovic et al., 2016, Mikolov et al., 2013b, Le and Mikolov, 2014, Vasile et al., 2016]). For users on the contrary, there is not much to be done and contextual information is often lacking. One can have access to device information, but not much more, which limits the interest of such algorithms. However there exist specific cases where using a content based algorithm is a good solution and is almost the only one: when user-item interaction is very rare but very meaningful, the catalog is quite small (hundreds of items) and a lot of contextual information are available on both users and items. Applications where all those criteria are met are the luxury and the automobile industries. For instance in automobile, brands have years of data on users with detailed contextual information, users have bought very few cars during those years but each purchase is very meaningful and lots of information are available on each car. On the other hand, in online applications collaborative filtering and hybrid methods usually provide better performances.

Collaborative Filtering

In Collaborative Filtering ([Schafer et al., 2007, Ekstrand et al., 2011]) no contextual information about users and items is assumed. Users are only defined by the items they interact with in the past and items by the users who interacted with them. This "definition" is much more precise and personalized than content-based methods since users are defined by their own personal tastes instead of descriptive features chosen by a human designer. The drawback is that the system needs to have collected enough feedback per user and item in order to learn information more precise than classic content-based models. Those more complex algorithms bring more precise prediction but also more complicated rules and are widely studied by the machine learning community. Another drawback on collaborative filtering is that the system learns hidden (or latent) descriptors that are much harder to understand by a human. On the contrary those latent descriptors create valuable embedding of users and items that can be used for other (marketing) applications like users and items clustering. Improving and generalizing state-of-the-art performance of collaborative filtering will be one of the main points of focus of this thesis.

Early methods, detailed in section 3.6.1, were based on explicit similarity measure between either users or items. Current methods are based on matrix factorization and rank minimisation using linear algebra or probabilistic models and will be detailed in section 3.6.1 and are one of the main points of focus of current recommender system research. Taking into account more and more types of information on users and items to improve performance is also a wide research area in the recommender systems community. This leads to the third familiy of hybrid methods.

Hybrid methods

Hybrid methods ([Burke, 2007]) goal is to mix together several models. Most of the time the goal of hybrid methods is to gather content-based methods and a collaborative filtering one to make one recommender system with the advantages of each. A simple solution consists in computing a weighted mean between a content-based model and a collaborative filtering one. The content-based model is denoted CB and the collaborative filtering model CF such that $CB(u, i)$ and $CF(u, i)$ returns the affinity prediction between user u and item i for each model. A hybrid model Hy can then be specified as follows:

$$Hy(u, i) = \alpha CB(u, i) + (1 - \alpha)CF(u, i) \tag{2.2}$$

where $\alpha \in (0, 1)$ is a tuning parameter that puts more or less weight to each model. But weighted hybrid methods are only one of the seven types of hybdrid recommender systems identified by the survey [Burke, 2002]. Other types are switching methods where for each prediction the system first chooses a recommendation component then applies it, mixed methods where recommendations of the different component are concatenated, features combination methods that consists in combining feature from multiple sources to create one algorithm, feature augmentation methods that create features that will serve as an input for a next model, cascade methods that prioritizes models according to a set of rules, and meta level methods where the output of a first model serves as the input of a next model. For instance, it is frequent that content-based models perform better than collaborative filtering on users and/or items with little feedback. Hence one would apply a cascade hybrid recommender system by prioritizing content based methods to predict user-item affinity with little feedback and collaborative filtering methdods otherwise. An automated solution consists in learning a new model on top of those two using any supervised learning techniques (for instance a decision tree, but in practice xgboost is often used) with the prediction of the previous models and some extra features (like the number of feedback provided by the user and on the item) as the input in order to learn to predict the true rating. This new model can now be considered as a meta-level hybrid recommender system that will distribute the weight among the different previous algorithms. Combining models effectively has proved with the Netflix challenge to lead to very effective recommender systems but also much more complicated ones since several models must be computed and combined before making a recommendation. As discussed in the previous section, deep learning techniques are now the state of the art to get better feature representation to be integrated into feature augmentation hybrid recommender systems ([Grbovic et al., 2016, Mikolov et al., 2013b, Le and Mikolov, 2014, Vasile et al., 2016]).

2.1.2 On predicting sets coherence

The main objective of predicting sets coherence is to suggest a new item for a user to add to her cart, that is to increase the value of users already well engaged with the brand but can also be used to generate set of items to be consumed together like a music playlist. This particular recommender systems problem belongs to the more general framework of session-based recommender systems which define all recommender systems designed at the session level. Session-based recommender systems have received more and more attention in the recent years, for instance in 2015 the RecSys Challenge was dedicated to session-based recommender systems. Unlike affinity based algorithms whose goal is to present one or several items that the user could buy independently, in basket completion the notion of set of items is crucial in order to present an item that is coherent with the items already in the basket since the goal is that the whole set will be purchased together.

Three main categories exist to tackle the basket completion issue: associative classification, collaborative filtering variations and Determinant Point Processes. Besides, more recently deep learning techniques have been developed using recurrent neural network ([Hidasi et al., 2015]).

Associative Classification

Associative Classification ([Agrawal et al., 1993, Hipp et al., 2000, Liu et al., 1998]) are simple methods that are based on association rules that compute sets supports to perform recommendation. The general idea is to count co-occurences of two sets using all past observed baskets in order to find the most frequent associations compared with the occurence of one of the two sets. Then, if this occurence ratio is high enough, the second set is considered to be a good recommendation when the first set is in a user basket. However, relying only on such a ratio may reduce to recommending only must popular items. Additional criteria are then defined to produce different rules that will define what to recommend.

Such approaches have several strong limits: the different criteria are arbitrarily defined and require a manual choice of thresholds, like the minimal value of the two sets co-occurence over one set occurence ratio, the impossibility to scale to large dataset and finally the necessity to update all previously computed rules whenever a new basket is observed, which can be very time consuming.

Adaptation of Collaborative Filtering

In order to scale to large item catalogs, a solution can consist in using collaborative filtering by considering each basket as one unique user. However in applications where baskets are very small (for instance in e-commerce it is very rare that people buy more than three products at the same time) the extreme sparsity of the matrix can prevent from learning correclty. Another solution consists in using the item latent factors obtained by the collaborative filtering algorithm in order to compute item-to-item similarities using for instance the cosinus between the two items. Then the most similar item is proposed to the user. The limit of such an approach is that recommending similar items may not be relevant since people are more likely to buy complementary items instead of two very similar ones. To that extent, people often add constraints to the recommendation like forcing the additional item to recommend to be in a different category than the initial item ([Teo et al., 2016]).

Determinantal Point Processes

More advanced methods based on Determinantal Point Processes (DPPs) ([Gartrell et al., 2017, Gartrell et al., 2016]) have shown very good accuracy and improved diversity in the recommendation. Determinantal Point Processes are probability measure over set of points defined by the determinant of a kernel matrix of the size of the item catalogs where each entry (i, j) of the kernel matrix encodes the similarity between product i and j. Based on past observed baskets, the kernel matrix of the DPP is learned, using for instance maximimization of the likelihood, and items to recommend are sampled from the obtained probability distribution. In order to scale to large item catalogs, [Gartrell et al., 2017, Gartrell et al., 2016] assumed a low rank constraint on the kernel matrix. This assumption proved to be both more effective to learn and predict and also produced more accurate recommendations.

2.1.3 Effective exploration-exploitation with bandit learning

When future observation directly depends on the output of the system, one must take care of the exploration vs exploitation dilemma in order to collect new information while also providing accurate recommendation. Besides in recommender systems applications, new users and new items arrive frequently and exploiring those users and items must be done appropriately. Recommender systems then leverage results and solutions from multi-arm bandit to solve this issue.

Multi-arm bandit

The exploration-exploitation dilemma is studied in the reinforcement learning theory and more specifically in the multi-arm bandit theory. Concretely the problem is to select the next action a (called arm, which in our case corresponds to an item) within a set of possible actions $\mathcal{A}$ in order to maximize the cumulative reward (i.e., the sum of the previously collected rewards) on the long term rather than maximize the immediate next step reward. Each arm is characterized by an unknown distribution and at each step the system observes the reward of the pulled arm.

A simple solution to introduce exploration in the system is to pull an arm at random a small fraction of the time and pull the best expected arm the rest of the time. This strategy is called ε-greedy where ε stands for the small fraction of time that random pulling is used, in practice $\varepsilon \approx 1\%$.

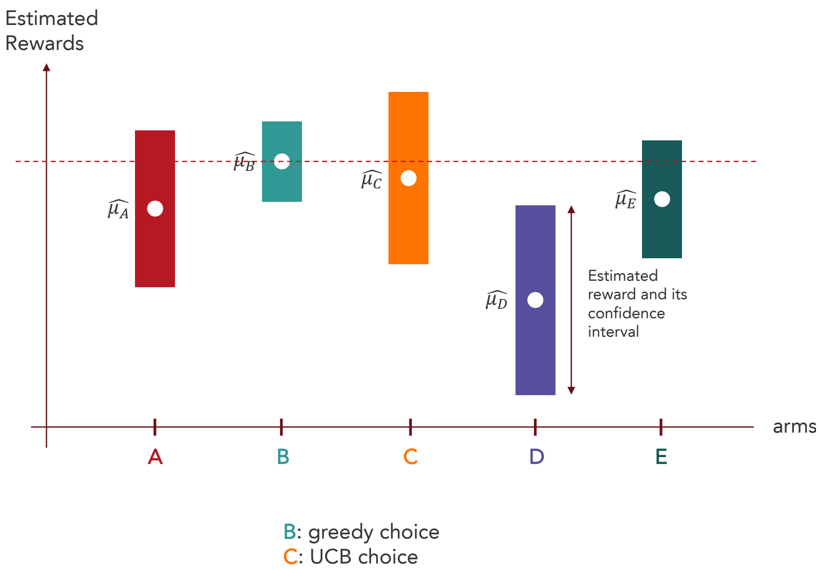
A more advanced solution is to use the Upper Confidence Bound (UCB) algorithm ([Auer, 2003]) that follows the optimism in face of uncertainty principle to select the next arm that increases the chance to select an arm with a lot of uncertainty. Intuitively when pulling the arm which in the best case scenario has the highest reward, there are two possible outcomes: either we did select the best arm and so we are good, either we did not and we gain knowledge on a good candidate. On the contrary a greedy strategy could stop pulling the best arm because of a few bad results in the first attempts. More formally, the UCB algorithm pulls at time t the arm I_t that follows equation 2.3.

$$I_t = \arg\max_{a \in \mathcal{A}} \hat{\mu}_{a,s} + \sqrt{\frac{3 \log t}{2s}} \tag{2.3}$$

where s is the number of times arm a has been pulled before time t and $\hat{\mu}_{a,s}$ is the empirical mean rewards observed by pulling arm a. This equation comes from a high-probability upper bound on the expected reward of each arm. See figure 2.2 for an illustration of UCB state at a given time step.

A similar approach can be developed in a bayesian framework using Thompson Sampling ([Kaufmann et al., 2012, Russo et al., 2017]). In such a framework one assumes that the reward obtained by pulling an arm follows a certain distribution with unknown parameters. At each time step, for each arm, a reward is sampled according to the current estimation of the distribution, and the arm with the highest

Figure 2.2: Upper Confidence Bound at a given timestep. In this situation the greedy solution would pull arm B because it has the highest estimated reward, and UCB would pull arm C because it has the highest upper bound.



sampled reward is pulled. Then its distributions parameters are updated following Bayes' rule, and one moves to the next step.

In this multi-arm bandit framework all arms are independent from each other, thus pulling one arm does not provide any information on the other arms performance which can be an issue when dealing with a lot of arms. Linear bandit allows to overcome this problem.

Linear bandit

The multi-arm bandit framework has then been extended to linear problem where each arm is defined by a vector $x_a \in \mathbb{R}^d$. After each pull the system observes the reward $r_a =< \theta^*, x_a > +\eta$ of the pulled arm, where $\theta^* \in \mathbb{R}^d$ is an unknown parameter common to all arms and η is a conditionally R-sub-Gaussian noise, see [Abbasi-yadkori et al., 2011], [Li et al., 2010b]. In such a model, pulling any arm gives information on all other arms thanks to the common parameter θ. Then similarly to UCB for multi-arm bandit, one can compute a confidence ellipsoid on θ in order to obtain a confidence interval on rewards and pick the arm with the highest upper bound.

Similarly to multi-arm bandit, one can use Bayesian ideas for linear bandit. [Agrawal and Goyal, 2013a] analyzed generalization of Thompson Sampling to linear bandit in algorithms called linear Thompson Sampling or Thompson Sampling for Contextual Bandits.

Bandit in recommender systems

In recommender systems application, linear bandit can be used in content based models where items are the arms of the bandit, and x_a represents the contextual information of a particular item. However learning a linear bandit per user can be very uneffective. To that end [Li et al., 2010c] propose a hybrid version of linear bandit where the reward is the aggregation of two linear terms: a first one with a contextual vector that jointly represents user and item weighted by an unknown coefficient common to all items and a second one with a contextual vector containing user features only weighted by an unknown coefficient different for each item. On the other hand, [Cesa-Bianchi et al., 2013] build upon linear bandit and assumed the existence of a user social graph in order to share information among each user linear bandit.

On some applications, it is frequent that the system must recommend several items at the same time. The user sequentially examines each item one after the other and clicks on the first one she likes. This problem, called cascading bandits, was studied by [Zong et al., 2016] using UCB and Thompson Samping like algorithms for multi-arm bandit and using linear UCB and linear Thompson Sampling like algorithms for linear bandit.

Using collaborative filtering, [Guillou et al., 2016] relied on ε-greedy method to perform exploration in their recommendations, whereas [Mary et al., 2014] adapt linear bandit to select item to recommend in applications where each item can be recommended several times to the same user.

2.2 Measure, performance and open datasets

2.2.1 Measure and performance

Evaluating the performance of a recommender system is not straightforward. Ideally one would like to A/B test the system, that is (version A) a certain portion of users see products chosen by the recommender system and others (version B) according to an other rule (random, popularity, a different recommender system), to assess which version is the best all other things being equal. But, as for any machine learning solution, one would rather have an offline evaluation protocol before testing any solution live. The classic supervised learning approach consists in selecting at random a fraction of the input data to train the model and to evaluate the model precision on the remaining data. Applying this protocol several times and computing the mean precision over all the different tests, gives a good idea on the model precision. This offline evaluation protocol, called K-cross validation, where K stands for the number of times this protocol is performed, is very classic in supervised learning, but must be executed carefully for recommender systems. In classic supervised learning settings, there are two possibilities (see figure 2.3) to perform K-cross validation. The first one consists in partitioning the input dataset into K non overlapping subsets, learning the model on $K-1$ subsets and computing

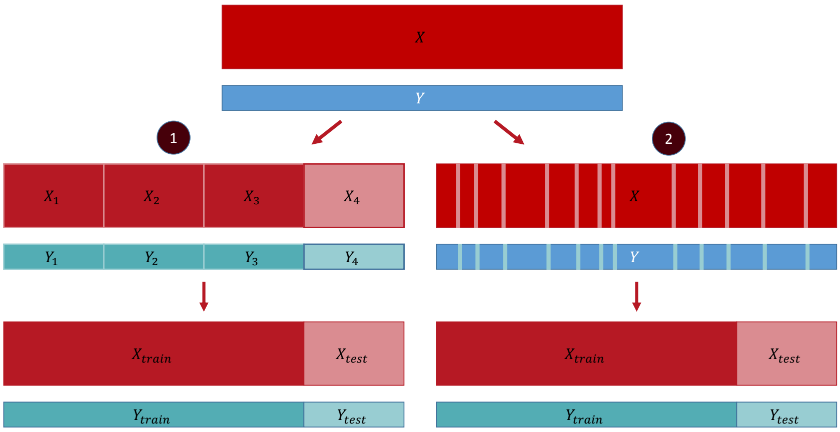


Figure 2.3: Illustration of cross validation. Left image, option 1, corresponds to K-fold cross validation when partitioning the input dataset (here with $K = 4$). The model is learned on all but one of the partitions that constitute X_{train} and Y_{train}, the model is then applied on $X_{test}(= X_4)$ and the error is measured between this prediction and $Y_{test}(= Y_4)$. The process is then repeated until each partition have played the role of the test set. The right image, option 2, corresponds to random sampling of the input dataset according to a given percentage. Unlike option 1 where the number of partitions defines the sampling percentage (if $K = 4$, the sampling is $75\%, 25\%$), with random sampling can choose K independently from the random sampling rate.

the difference between the true value in the left subset and the predicted ones. A second option is to learn the model on $x\%$ of the input chosen at random and measure the performance on the left $(100 - x)\%$. However directly applying one of those two solutions can lead to some issues for recommender system offline evaluation. Indeed, when selecting input at random, some users (resp. items) may never be selected in the training set, thus making the prediction infeasible for those users (resp. items). A possibility could be to remove users and items with little feedback from the whole dataset. This option is often criticized because it is considered as removing hard entries and thus keeping only the easy ones. However, it depends on the goal of the recommender system: if the main goal is to improve accuracy on users and items with sufficient knowledge, that is finding the best model on "warm" users and items, it makes sense; on the contrary if the main goal is to find the best global model it does not. A solution consists in selecting for each user $x\%$ of its feedback at random and at least one (some users may have only one feedback in the dataset) and likewise for items. The union of this sub-selection for each user and item forms the final training set and the remaining values define the test set. This protocol, called stratified cross validation, ensures to have feedback for each user and item. We will refer to this version as *stratified cross validation*. However this selection is still not perfect since there could exist time bias in the provided feedback: users' future ratings can depend on previous ratings, thus having ratings in the training set that are posterior to ratings in the test set could lead to bias the model and to a poor generalization performance in real applications. Thus a more appropriate splitting protocol would be to remove last rated items for each user. Once the training set and test set are obtained and the model learned, an error measure has to be chosen to define the precision. At the beginning, recommender systems used to be evaluated according to the Root Mean Squared Error (**RMSE**), as it is often the most adequate performance measure in supervised learning. Let note $\mathcal{T}$ the set of prediction to make, the $RMSE$ is

$$RMSE \doteq \frac{1}{|\mathcal{T}|} \sum_{i \in \mathcal{T}} (y_i - \hat{y}_i)^2 \tag{2.4}$$

where y_i is the true value of the i^{th} observation and $\hat{y}_i$ is the prediction according to the learned model.

However in most applications the system will choose only K items to recommend, and the $RMSE$ does not gives guarantees on the relevance on those K recommended items. In such application the real goal is to preserve the preference ordering of each user and especially for the items that the user likes the most. Indeed, two models with the same $RMSE$ can have (relatively) different items ordering and thus two really different performances in practice. Suppose that the true ratings for a user are $3; 4; 1; 5$. If model 1 predicts respectively $3.4; 3.3; 1; 4$ and model 2 predicts $2.6, 3.3; 2; 5$, both models have a $RMSE = 0.4125$ but only model 2 preserved the right order and thus will provide the best possible top-K recommendation. There exists lots of more suitable metrics for such applications that compute the ordering of the prediction instead of the value predicted, like Kendall's τ, Spearman correlation and other many classic correlation measures. In the recommender systems field a classic measure is the **rank measure** that weights the ordering of the predictions according to the true feedback:

$$\overline{\rho} \doteq \frac{\sum_{u,i \in \mathcal{T}} r_{u,i}\rho_{u,i}}{\sum_{u,i} r_{u,i}}, \tag{2.5}$$

where $\rho_{u,i}$ denotes the percentile-ranking of the item i of all items previously seen by user u ordered according to the predicted values from the most suitable item to the least suitable one, and $r_{u,i}$ the true feedback. For instance, if item i is predicted to be the most (resp. the least) suitable for user u, then $\rho_{u,i} = 0\%$ (resp. $\rho_{u,i} = 100\%$). If the predictions are made at random, the expected rank-measure is 50%. Low values of $\overline{\rho}$ indicate that most of highly seen items have been predicted to be the most suitable and thus correspond to a good recommendation. On the other hand, $\overline{\rho}$ above 50% means that the algorithm is no better than a random strategy.

Another popular evaluation measure, commonly used in information retrieval, is the **DCG@k** for Discounted Cumulative Gain at k:

$$DCG_u@k \doteq \sum_{i=1}^{k} \frac{2^{rel_{u,i}-1}}{\log_2(i+1)} \tag{2.6}$$

where $rel_{u,i}$ is the relevance of the i^{th} recommended item for user u, that is in our case the true feedback. This function measures the relevance of the top K recommended items for a user, and then only focuses on the top of the list. This measure is particularly interesting for recommender systems since the user interacts with top products, and thus we want to know how relevant top predictions are, no matter what the other ones are. However this measure is not consistent through users that may have different test set lengths or rating behaviours. We thus use the **NDCG@k**, the Normalize Discounted Cumulative Gain, where the $DCG@k$ is divided by the best value of $DCG@k$ that is when the items are in the best possible order, also called the Ideal DCG (IDCG):

$$NDCG_u@k = \frac{DCG_u@k}{IDCG_u@k} \tag{2.7}$$

Computing the mean $NDCG_u@k$ over all users gives the global performance of the model: $NDCG@k = mean_u NDCG_u@k$.

In the particular case of $K = 1$ two others measure are often used: the Mean Percentile Rank (**MPR**) and the Precision at k (**precision@k**), where here k denotes the top k predictions. The MPR measure sorts all possible items to recommend to a user u according to their predicted scores and output the percentile rank (PR) of the item j_u, where j_u is the item the user actually consumed, among candidates items $\mathcal{C}$ - that is items that have not been consumed by the user - that the user actually consumed:

$$PR_{j_u} = \frac{\sum_{i \in \mathcal{C}} \mathbb{I}(\hat{r}_{u,j_u} \geq \hat{r}_{u,i})}{|C|} \times 100\%, \tag{2.8}$$

Taking the average over all users in the set set $\mathcal{T}$ gives the mean percentile rank

$$MPR = \frac{\sum_{u \in \mathcal{T}} PR_{u,j_u}}{|\mathcal{T}|} \tag{2.9}$$

A MPR of 100% means that the actual consumed item always gets the highest score, and a MPR of 50% corresponds to a random model. Finally, the precision@k counts how many times the target item was in the top K predicted ones. Let note $rank_{u,j_u}$ the rank of the target item j_u of user u, thus

$$precision@k = \frac{\sum_{u \in \mathcal{T}} \mathbb{I}(rank_{u,j_u} \leq K)}{\sum_{j_u} 1} \times 100\% \tag{2.10}$$

A common limit to all offline evaluation mesures is that they can only be computed on observed ratings which introduces a bias because ratings are missing not a random. Indeed in most available datasets, available ratings are ratings that users deliberately decided to rate and have a different distribution from missing values. [Steck, 2013] worked more in details on this problem and thanks to Yahoo! Music data that explicitly asked users to rate some songs at random and emphasized this bias: the average rating of randomly selected songs is 1.8 whereas the average rating of deliberately rated ones is 2.9. Consequently a model with a low test set RMSE may not achieve a low RMSE on the whole possibilities. More problematic, the system that achieves the best performance on observed ratings may not be the one that achieves the best performance on all the possibilities and thus should not be used in pratice (see Table 1 of [Steck, 2013]).

The metrics introduced above only measured the current precision/performance of the model according to the current state of users and can be suitable to measure the performance of the system in a particular setting, for instance on little engaged users. However in some applications one can be interested in the dynamic of the algorithm that is its ability to produce recommendations that both please the users and collect valuable feedback for future prediction. For instance for a cold user, the systems may provide poor recommendations at the beginning but acquire very informative feedback that will produce good recommendations on the long term. This can be measured by the cumulative reward that is the sum of feedback received after each recommendation up to time t:

$$cum\_reward_u(T)_t = \sum_{k=t+1}^{T} r_{u,i(t)} \tag{2.11}$$

where $i(t)$ is the t^{th} recommended item to the user. In practice measuring the cumulative reward on offline data is complicated. Indeed one must have access to the rating that the user in the test would have given to the item recommended by the system, which may not be available. The first trick consists in updating the model if and only if the recommended item for a given user was indeed available in the dataset. This strategy called *replay* has been studied by [Li et al., 2010a] that proved that it was an unbiased offline evaluation when the data stream was infinite and when the logging policy, that is the way data was actually collected, picked arm at random. The second trick consists in keeping in the test set only users with the highest amount of feedback in order to reduce the risk of not having the observation. However this second trick can create bias since very engaged users may have a very different behaviour from the rest of the world. Another direction consists in building a simulator. The idea is to first estimate the reward of each arm according to all available data. Then those estimates are supposed to be the actual reward (plus some noise) that one would get when pulling each arm. Then, to test the policy, the system starts with an empty history and picks an arm according to the policy to test. A reward is then sampled according to the previously computed simulator and the system is updated. In practice, constructing a simulator can be difficult and requires a large history in order to have an unbiased estimator.

All those error metrics only measure some definition of the accuracy of the recommender system, however in practice one may be interested in other performances. For instance the diversity of the recommended items introduced by the system can be a desired criterion as well as the novelty of the system that is the capacity of the system to recommend items that are new to a user, or the catalog coverage that measures the proportion of the catalog that is actualy recommended, and the serendipity that defines the fact of being both useful and unexpected. Obviously it is impossible to simultaneously maximize accuracy, diversity, novelty, coverage and serendipity and each measure is not necessarily positively correlated to the real online performance of the system. [Maksai et al., 2015] proposed a model to link the online performance to all those different performance measures in order to help decision makers to decide what model to test online.

To sum up, in recommender systems experimental protocol definition plays an important role in the model evaluation from the train-test split to the choice of the performance measure and must be done carefully and according to the desired application goal.

2.2.2 Available datasets

A limited number of open datasets are available for everybody to evaluate and compare their own models with the state of the art on a common basis. These different datasets allow to test different objectives and assumptions. Here is a non-exhaustive list of some useful ones.

- MovieLens [Harper and Konstan, 2015]: There exist five differents MovieLens datasets containing either 100k, 1M, $10M$, $20M$ and more than 20M (frequently updated) movie ratings. Each dataset contains 4-tuple (user id, item id, rating, timestamp) as well as user and item contextual information like gender and genres.

- LAST.FM [Celma, 2010]: Several music datasets are available with Last.fm. A first sparse one is formed of 1k users listening history with timestamp, artist and song resulting in around 19k rows, and a second denser one of 360k users resulting in around 17k rows. An API is also available to obtain extra information like song tags.

- YAHOO [yah, 2013]: Eight datasets containing ratings are available on YAHOO's website. Four datasets contain ratings on music on different time periods and sizes (from several thousand to a few millions of users). One dataset contains movies ratings, one bookmarked website on del.icio.us website and two datasets contain clicks on Yahoo! front page today module.

- YOOCHOOSE [Ben-Shimon et al., 2015]: This dataset was realeased during RecSys 2015 Challenge and is formed of two files extracted gathering event from an unknown e-commerce website of more than 9M sessions. The first one contains click events with a session id, a timestamp an item id and an item category. The second one contains buy events that is price, timestamp and quantity of the bought item id of a given session id.

- NETFLIX [Bennett et al., 2007]: Released for the Netflix Challenge, this dataset is no longer public since the closure of the challenge. More information on this challenge is provided in the following section.

- AMAZON BABY REGISTRIES [Gillenwater et al., 2014]: is a public dataset consisting of $110,006$ registries of 15 disjoints registry categories like diaper, apparel or feedings, with 100 items per category. The categories are disjoints meaning that a basket with products of one category cannot contain products of another category. The average basket size in one category is between 2 and 3 depending on the category.

- BELGIAN RETAIL SUPERMARKET [Brijs et al., 1999]: is a public dataset that contains $88,163$ sets of items that have been purchased together, with a catalog of $16,470$ unique items. Each basket contains 9.6 items on average.

- UK RETAIL [Chen et al., 2012]: is a public dataset that contains $22,034$ sets of items that have been purchased together, among a catalog of $4,070$ unique items. This dataset contains transactions from a non-store online retail company that primarily sells unique all-occasion gifts, and many customers are wholesalers. Each basket contains 18.5 items on average, with some very large baskets.

- INSTACART [ins, 2017]: is, as far as we know, the only public dataset that contains the order in which the product have been added to someone basket.

It is formed of three datasets containg online grocery shopping of more than $200,000$ Instacart users: a "train" dataset with around $131,000$ baskets, the last basket of each user, a "prior" dataset with all past baskets for each user (around 3.2 millions baskets) and a "test" dataset with around $75,000$ baskets reserved for machine learning competitions. All those orders are made on approximately $50,000$ products and each basket contains 10.5 products on average.

Learning Affinities From Multiple Type of Feedbacks

As seen previously, recommender systems rely on past feedbacks to provide personalized recommendations. The feedbacks may take different forms, such as the rating of a movie, or the number of times a user listened to the songs of a given music band. Improving precision and theoretical guarantees of recommender systems has been widely studied in cases in which users can provide a single type of feedback on items. Nonetheless, in some situations, the user can perform several actions on each item, resulting in a multidimensional feedback (e.g., the user of an e-commerce website can either click on a product, add the product to her cart or buy it). However state-of-the-art algorithms rely on matrix factorization where each row of the matrix represents a user, each column an item and each entry is either empty or contains the corresponding feedback. In the case of multidimensional feedback, such algorithms can not be applied any more, unless the problem is reduced to a series of multiple independent problems, thus loosing the correlation between the different actions. The most suitable approach is to use a tensor approach to learn all dimensions of the feedback simultaneously. In this section, we propose a specific instance of tensor completion and we show how it can be heavily parallelized over both the dimensions (i.e., items, users, actions) and within each dimension (i.e., each item separately). We validated the proposed method both in terms of prediction accuracy and scalability to large datasets.

3.1 Introduction

The aim of personalized recommender systems (RSs) is to discover the preferences of each user and to predict which items, among those available in the system, she likes the most. The collaborative filtering approach [Hu et al., 2008, Zhou et al., 2008, Koren et al., 2009, Mackey et al., 2011, Salakhutdinov and Mnih, 2008] relies on user-item interactions to infer implicit descriptions suitable for prediction. Since its success in the Netflix Prize competition [Bennett et al., 2007], one of the most popular collaborative-filtering method is based on matrix factorization techniques for latent factor models. In this approach, the recommender system problem is reduced to the problem of completing a highly sparse user/item matrix (i.e., each entry records the feedback of the user for a specific item) under low-rank assumptions, where rows refer to the users and columns refer to the items available in the system. In general, the feedback can be either explicit (e.g., a rating on a scale from 1 to 5) or implicit (e.g., number of views, number of times a song is listened to, number of clicks).

Despite its success, matrix factorization cannot always capture the full complexity of the problem. In several domains, users can perform several actions on the items. For instance, in e-commerce applications, while the user-item affinity is often predicted on the basis of the number of times the user clicked on the item, the action one would like to predict is whether the user will actually add the product to her cart and/or buy it. Unfortunately, these actions are much sparser than the clicks, thus far more difficult to predict, and they are not necessarily positively correlated with them, so predicting clicks will not make an effective recommender system. As a result, applying matrix factorization only on the click action may result in poor recommendation for the actual actions of interest (i.e., buy). Indeed, a high number of clicks on a product may not mean that the user is really interested in the product but rather that she is hesitant. Another scenario in which matrix factorization is not satisfactory is for recommender systems where the recommendation needs to be related, e.g., with a specific moment of the day or the season. For instance, previous studies on music recommendation [Baltrunas and Amatriain, 2009] have shown that people listen to different artists depending on the moment of the day or of the week (while working, driving, resting at home). Again, ignoring this dimension and applying matrix factorization directly on the user-item matrix, may severely affect the final performance of the system. Of course all those actions can also happen in the same time, one may want to take into account the "add to cart" based on the day of week or the season of the year resulting in a three dimensional feedback.

An effective approach in case of multidimensional feedback is to use tensor factorization techniques, which extend matrix factorization by taking into consideration all actions/contexts at the same time. In particular, the user-item matrix is expanded by adding one or more modes, which contains the additional type of actions available, like the number of clicks, add to cart and purchase of each item. Tensor factorization is considerably more complicated than matrix factorization and it requires extending previous methods. [Symeonidis et al., 2008] rely on a generalization of the singular value decomposition for tensors called *higher-order singular value decomposition* (HOSVD) introduced by [Lathauwer et al., 2000], and it is part of the general family of Tucker decomposition. In HOSVD, the n-order tensor is unfolded into n matrices on which classic SVD is applied. Its decomposition is then used to reconstruct the full tensor factorization. An alternative approach is to perform tensor decomposition using PARAFAC (parallel factor analysis). For instance, [Hu et al., 2013] performed cross-domain recommendation using an alternating least squares algorithm on the PARAFAC model called CP-ALS-R (PARAFAC is also called CANDECOMP, that is what the CP stands for here). However, recommendation problems always come with missing values (otherwise there would be nothing to learn) and this problem has not been taken into account neither in CP-ALS-R nor HOSVD. This problem has been studied for tensor decomposition for instance in the chemometric field for the PARAFAC decomposition [Acar et al., 2010, Tomasi and Bro, 2005] and the Tucker decomposition [Filipović and Jukić, 2013]. Finally, [Romera-Paredes and Pontil, 2013] studied the tensor completion problem not with tensor factorization but working on the trace norm regularization and introduced a new convex relaxation method for tensor completion. Unfortunately, most of these techniques suffer from different shortcomings when applied to recommender systems: **1)** they may not properly manage missing values (i.e., by filling unknown feedback with arbitrary values), **2)** they do not allow for modes with latent factors of different sizes (i.e., constraining to a single dimension for all factors), **3)** they do not scale to large tensors (both in size and number of modes) because of algorithm complexity and storage issues.

Contributions. In this chapter, we focus on an adaptation of the alternating least squares approach used in matrix factorization. In dealing with matrix factorization for recommender systems, alternating least squares with λ regularization (ALS-WR) introduced by [Zhou et al., 2008] proved to perform much better than SVD. While both approaches assume a low-rank constraint on the matrix, they differ in the way they deal with unknown values. While SVD implicitly sets all unknown values to 0, the ALS-WR minimizes the ℓ_2-norm error between the true feedback and the factorized matrix only for the available entries, so no assumption is made on the unobserved values. If all unknown values are set to 0, meaning that unknown feedback is equivalent to a negative feedback, the best factorization of the input matrix into two rank-k matrices in the Frobenious norm sense is given by the SVD algorithm. However, when the sparsity level of the input matrix is very high, such an hypothesis introduces a large bias which prevents the SVD algorithm to find a good factorization. Besides, the ALS-WR also introduces a penalty term based on Tikhonov regularization which penalizes large parameters in order to prevent from overfitting. This regularization is further weighted based on the number of feedback provided by a user and the number of feedback received by an item. This simple modification showed much better performance in practice since it puts different regularizations on users and items based on the size of their history in order to have a stronger regularization on very popular items and very engaged users that are more likely to be overfitted. It also helps generalisation since the regularization parameter will automatically be adjusted for each user as soon as she provides a feedback. Building on this observation, we study a method called HOALS, for Higher Order Alternating Least Squares, that leverages alternating least-squares to perform tensor factorization. Aside from being better designed for recommender systems, we

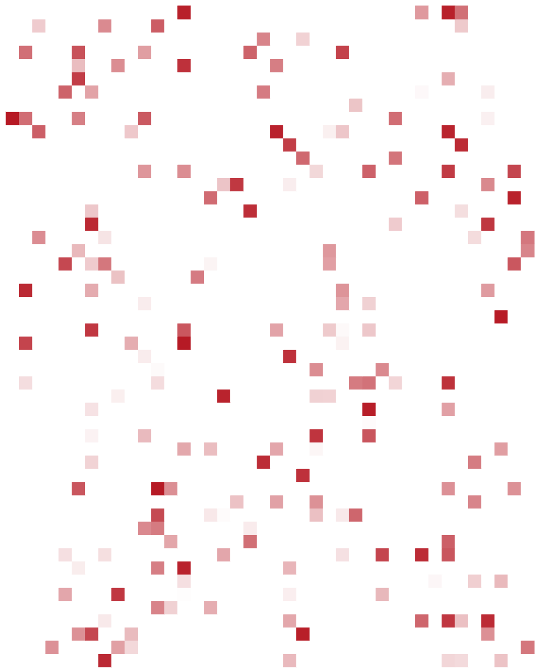


Figure 3.1: Example of the input matrix in recommender system with a sparsity level of 90%. Each row represents a user, each column an item and each square the feedback. White squares are missing value, the darker the higher the affinity is.

prove that HOALS can be easily parallelized both over tensor modes and within each dimension, thus allowing it to be applied to large recommender systems. Finally, we showed that the HOALS formulation allows us to manage implicit feedback along the line of [Hu et al., 2008]. To the best of our knowledge, HOALS is the only tensor factorization model that allows to perform tensor factorization for implicit feedback, that is able to deal with missing data and is scalable to large problems.

3.2 Background

3.2.1 Recommender Systems as matrix completion

State-of-the-art methods focus on the problem of filling the sparse input matrix $R \in \mathbb{R}^{n \times p}$, with n the number of users and p the number of items, see figure 3.1, according to the known entries of the matrix. Each row represents a user $u \in \mathcal{U} = (1, \ldots, n) \doteq [n]$, each column an item $i \in \mathcal{I} = (1, \ldots, p) \doteq [p]$ and each entry $r_{u,i}$, if available, the user-item affinity. We denote $\mathcal{D} = \{(u, i) \text{ such that } r_{u,i} \text{ is known }\} \subset \mathcal{U} \times \mathcal{I}$. In practice the matrix is highly sparse, for instance the matrix on the Netflix challenge has only 1.2% available entries. The matrix completion problem in recommender system is particularly difficult because unlike image completion one

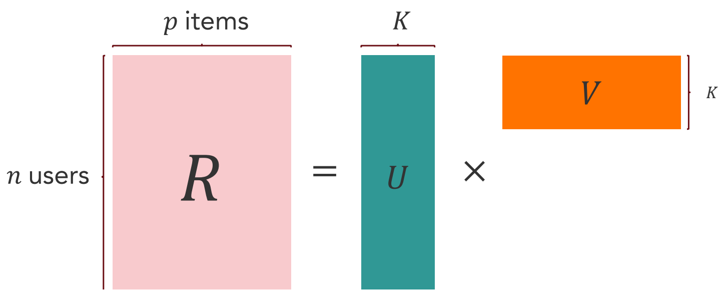


Figure 3.2: Matrix Factorization.

can not use local information since the rows and columns ordering is random and available input is not uniformly distributed. Indeed some items are very popular and will have been rated by many users when some others will be almost never rated which can results in having some blank area very difficult to fill appropriately. This behaviour is called MNAR for Missing Not At Random. Learning is then performed by minimizing some distance between the model predictions and the available entries. However some structure assumptions on the matrix are required otherwise any matrix that has the exact same values at the known coordinates and random values elsewhere will have exactly zero error but no good completion can be expected. The more realistic the regularization is, the better the performance of the algorithm. A usual assumption is that the input matrix is low rank, which means that there exist some correlation among users and among items. Consequently the matrix can be factorized into two sub matrices U and V such that $R = UV^T$ where $U \in \mathbb{R}^{n\times K}$ and $V \in \mathbb{R}^{p\times K}$ has as many columns as the assumed rank K of R, see figure 3.2. U contains users latent descriptors and V items latent descriptors. The low rank decomposition has the advantage to reduce the memory since $(n+p)K$ (with $K \ll n, p$) values need to be stocked instead of np. Indeed the full matrix has never to be computed, since to decide what to recommend to one user, only its latent factors and the items latent factors are required. The problem then falls back to learning the two matrices U and V. Historically, matrix factorisation is learned through Singular Value Decomposition (SVD) where only largest eigenvalues are kept, which also allows to reduce the noise that can lie in the small eigenvalues. However such techniques are not really reliable for applications with sparse matrices. Consequently, the input matrix have to be filled with arbitrary values, often zero or the mean of the available values, before applying SVD, which can lead to poor performance. A better approach consists in learning the factorization using only values in $\mathcal{D}$ like ALS-WR.

> ALS-WR [Zhou et al., 2008] - Alternating Least Square with Weighted-λ-regularization is one of the most popular collaborative filtering algorithm thanks to its high scalability and the available machine learning libraries that imple-

ment this method like *mahout* or *spark*. ALS-WR relies on two main ideas. First, since some users provided more feedback than other and likewise some items received more feedback than others, the regularization should not be the same for each user and item but should rather depend on this number of feedback. With n_u and n_i respectively the number of feedback provided by user u and provided for item i, the objective function is then:

$$\min_{U \in \mathbb{R}^{n \times K}, V \in \mathbb{R}^{n \times K}} \sum_{u,i \in \mathcal{D}} \left(r_{ui} - U_u V_i^T \right)^2 + \lambda \left(n_u ||U_u||^2 + n_i ||V_i||^2 \right) \tag{3.1}$$

We see that users with lots of feedback will have a stronger regularization than others. The second idea of ALS-WR is as the name suggest, to use an alternating algorithm to solve 3.1. We first fix V with small random numbers and solve the objective function on U. The problem can be rewritten as n independent penalized linear regressions (one for each user) and can be solved in parallel. For instance, once V is fixed, the minimization problem to find the user u latent factor is

$$\min_{U_u \in \mathbb{R}^K} \sum_{i \in I_u} \left(r_{ui} - U_u V_i^T \right)^2 + \lambda n_u ||U_u||^2 \tag{3.2}$$

thus

$$U_u = (V_{I_u}^T V_{I_u} + \lambda n_u Id_K)^{-1} V_{I_u}^T R_{u,I_u} \tag{3.3}$$

Then we fix U to solve V which is again p independent penalized linear regressions, one for each item. The convergence is usually obtained within 10 or 20 steps and thanks to parallelization is easily scalable to very large problem. [Hu et al., 2008] adapt this algorithm to implicit feedback that can take very large values. Such large values can cause learning issues when minimizing the square residuals since the model will only try to fit those values and completely disregard other part of the matrix. **implicit** ALS-WR solves this problem by changing the objective function to fit the value 1 when the feedback is available, which indicate that the user likes the item, with a confidence weight proportional to the original feedback:

$$\min_{U \in \mathbb{R}^{n \times K}, V \in \mathbb{R}^{n \times K}} (1 + \alpha r_{ui}) \sum_{u,i \in \mathcal{D}} \left(1 - U_u V_i^T \right)^2 + \lambda \left(n_u ||U_u||^2 + n_i ||V_i||^2 \right) \tag{3.4}$$

where $\alpha \in \mathbb{R}$ is a scaling weight. To prevent from fitting only positives value equal to 1, one can sample a small percentage of missing values and treat them as 0. Thus let note $l_{ui} = 1$ if r_{ui} is known, and 0 only $\varepsilon\%$ of the time when r_{ui} is missing. We then defined $\mathcal{D}^\varepsilon = \{(u,i), l_{ui} \geq 0\}$ such that $\mathcal{D} \subset \mathcal{D}^\varepsilon \subset \mathcal{U} \times \mathcal{I}$.

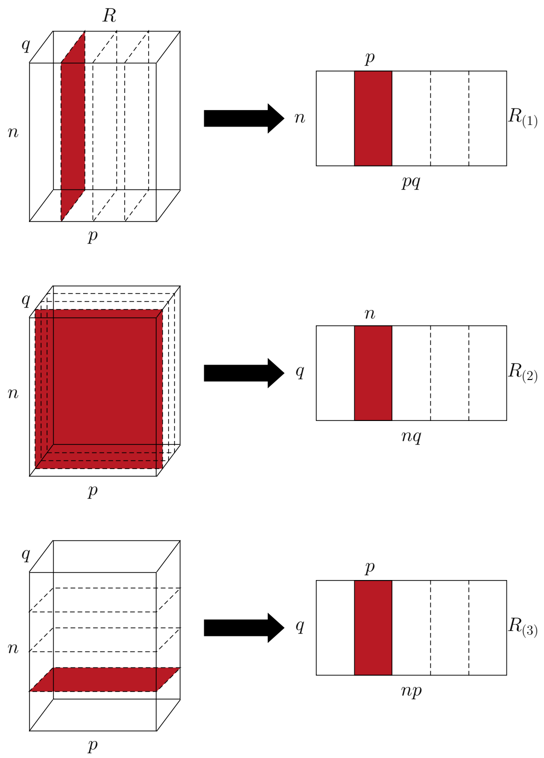


Figure 3.3: Mode matrix unfolding of a 3-order tensor. For instance, X_1 is the result of stacking all the lateral slices of the tensor.

We end up with the slightly modified objective function:

$$\min_{U\in\mathbb{R}^{n\times K}, V\in\mathbb{R}^{n\times K}} (1+\alpha r_{ui}) \sum_{u,i\in\mathcal{D}^{\varepsilon}} \left(l_{ui} - U_u V_i^T\right)^2 + \lambda \left(n_u ||U_u||^2 + n_i ||V_i||^2\right) \quad (3.5)$$

3.2.2 Recommender systems as tensor completion.

Here we extend the definition of a recommender system with an extra dimension $\mathcal{A} = (1, \dots, q) \doteq [q]$, the set of actions the user can take over items (e.g., a binary value such as click, add to cart (ATC), purchase, or a number as in the case of a rating), such that a recommender system is now defined as a tuple $\langle \mathcal{U}, \mathcal{I}, \mathcal{A} \rangle$. In the following, we only consider one action dimension, but in general other dimensions could be available. For instance, we could consider the rating attributed by a user to the food, to the service and to the decoration of a restaurant in a specific season (i.e., an additional dimension of interest) resulting in a 4-order tensor. We will use u, i, and a to index the elements in the sets $\mathcal{U}$, $\mathcal{I}$, and $\mathcal{A}$ respectively. Each feedback is a scalar value $r_{uia} \in \mathbb{R}$, which denotes the feedback provided by user u on item i through action a. For instance, r_{uia} may be the rating provided by user u to the feature a of a restaurant i. Once organized along the sets $\mathcal{U}$, $\mathcal{I}$, $\mathcal{A}$, all the possible feedback take the form of a tensor (i.e., a multidimensional array) $\mathbf{R}$ of dimension $n \times p \times q$. Nonetheless, only a limited number of feedback is available to the system. We denote by $\mathcal{D} \subseteq \mathcal{U} \times \mathcal{I} \times \mathcal{A}$ the set of triples (user, item, action) for which a feedback is available. Starting from the feedback available for the tuples in $\mathcal{D}$, the objective of a recommender system is to complete the tensor $\mathbf{R}$ with all missing entries and propose the best recommendations for each user. While in the case of one single action, for any user u the recommender system simply recommends the item i with the largest value r_{ui}, when multiple actions are available, the recommender system should combine different values r_{uia} to find a global index of preference. For instance, a recommender system can choose to recommend the restaurant with the best average rating over all features (i.e., average of ratings for the service, the food, and the decoration) or one can introduce a filtering on the rating of different actions (e.g., filter out restaurants with decoration rated less than 7/10) and then pick one according to a chosen feature (e.g., the one with the best rated food) or again compute a weighted average based on feature relevance for each user. In other cases, only one action is actually interesting, for instance buy or not, whereas other actions are only here to provide context and additional information on user-item affinity. Thus, recomendations are done by sorting the desired action prediction.

Tensor notation. In the following, we use bold upper case letters, such as $\mathbf{X}$, to denote tensors, whereas matrices are denoted by upper case letters such as X. The pseudo-inverse of a matrix is denoted by X^*. The entry of a third-order tensor is written with a lower case (e.g., x_{i_1,i_2,i_3}) where first, second and third indices represent respectively row, column and depth (similar for matrices, x_{i_1,i_2} denotes the element in row i_1 and column i_2). In the general n-order tensor we may use *fiber* instead of row, column and depth, to denote one dimensional part of the tensor, which can not find equivalent name when n is greater than 3. A particular fiber is called mode-i. For instance, for a two-order tensor the column corresponds to mode-1 and row to mode-2. Moreover a two dimensional part of the tensor is called a *slice*. In the special case of matrices, we denote the uth row of X by X_u. We recall some basic definitions and operations on tensors. Let $\mathbf{X} \in \mathbb{R}^{I_1 \times \cdots \times I_N}$ be a N-dimensional tensor, we define the mode-n product of tensor $\mathbf{X}$ by a matrix $U \in \mathbb{R}^{J_n \times I_n}$, denoted $\mathbf{X} \times_n U$, as the tensor of dimension $(I_1 \times \cdots \times I_{n-1} \times J_n \times I_{n+1} \times \cdots \times I_N)$ obtained as

Definition 1 (mode-n product of tensor).

$$\left[\boldsymbol{X} \times_n U\right]_{i_1 \cdots i_{n-1} j_n i_{n+1} \cdots i_N} = \sum_{i_n=1}^{I_n} X_{i_1 \cdots i_n \cdots i_N} U_{j_n, i_n}.$$

The mode-n product is commutative when applied in distinct modes, i.e. for $m \neq n$,

$$(\mathbf{X} \times_n A) \times_m B = (\mathbf{X} \times_m B) \times_n A.$$

while for $m = n$ we have,

$$(\mathbf{X} \times_n A) \times_n B = \mathbf{X} \times_n (BA).$$

Another important operation is the matrix unfolding of the tensor. There are N different ways to unfold a N-order tensor into a matrix called 1-mode, 2-mode,..., N-mode matrix unfolding. For instance for $N = 3$, one can stack the row, the column, or the depth as shown in Figure 3.3. In general, the d-mode matrix unfolding results in a matrix $X_{(d)} \in \mathbb{R}^{I_d \times I_1 \cdots I_{d-1} I_{d+1} \cdots I_N}$.

Tensor factorization. Unlike matrix factorization, tensor decomposition does not have a unique formulation and many different methods exist to recover a tensor as a combination of multiple factors (see e.g., [Kolda and Bader, 2009]). The two principals tensor decomposition formulations are the CANDECOMP-PARAFAC (*CP* for short) decomposition and the Tucker decomposition. The CP decomposition factorizes a n-order tensor into a sum of rank-one tensors. More precisely, a three-order tensor $\mathbf{X} \in \mathbb{R}^{N \times M \times P}$ is decomposed as

Definition 2 (CP decomposition).

$$\boldsymbol{X} = [A, B, C] = \sum_{k=1}^{K} A_{:,k} \otimes B_{:,k} \otimes C_{:,k},$$

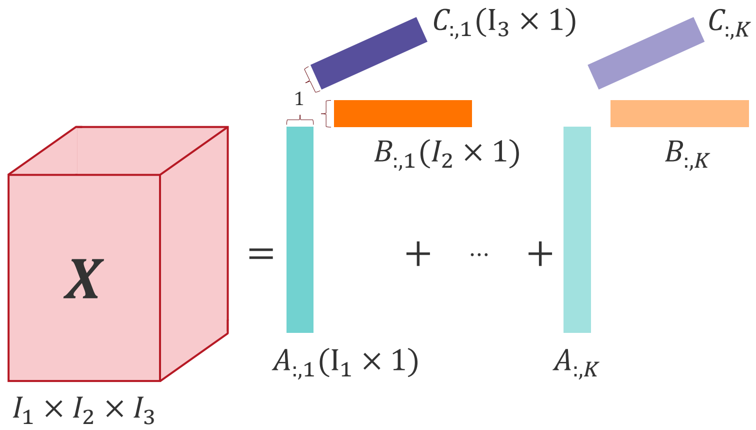
where $A \in \mathbb{R}^{N \times K}$, $B \in \mathbb{R}^{M \times K}$, $C \in \mathbb{R}^{P \times K}$, K an arbitrarily chosen dimension, and $\otimes$ denotes the outer product. As a result, the entries of the tensor can be computed as $\mathbf{X}_{ijp} = \sum_{k=1}^{K} A_{i,k} B_{j,k} C_{p,k}$. The minimal value of K for which such a decomposition exists defines the rank of the tensor. On the contrary, the Tucker decomposition assumes that the tensor $\mathbf{X}$ can be factorized as

Definition 3 (Tucker decomposition).

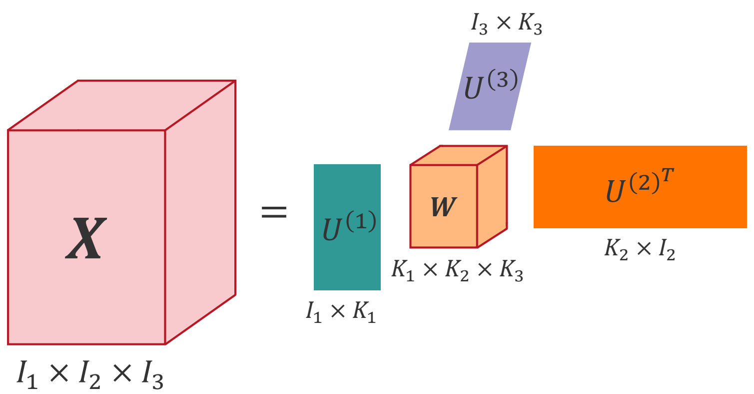
$$\boldsymbol{X} = \boldsymbol{W} \times_1 U^{(1)} \times_2 U^{(2)} \times_3 U^{(3)}$$

where $U^{(d)}$ contains orthonormal vectors called the d-mode singular values and $\mathbf{W}$ is the core tensor. In general, for any $d \in \{1, 2, 3\}$ the matrices are $U^{(d)} \in \mathbb{R}^{I_d \times K_d}$ and the core tensor is $\mathbf{W} \in \mathbb{R}^{K_1 \times K_2 \times K_3}$, where K_1, K_2, and K_3 are dimensions arbitrarily chosen.

In general, computing the CP and the Tucker decompositions is a computationally challenging problem. In the next section we present an efficient parallel algorithm, called *higher order alternating least-squares* (HOALS), to perform the general tensor factorisation. In the following section we compare HOALS to other state of the art methods.



(a) CANDECOMP-PARAFAC decomposition



(b) Tucker Decomposition

Figure 3.4: Factorization of 3-order tensor.

Algorithm 1 Direct-HOALS for Tucker factorization.

Input: dataset $\mathcal{D}$, desired ranks $\{K_d\}_{d=1}^{3}$
Construct the initial (sparse) 3-order tensor $\mathbf{R}$ from $\mathcal{D}$
Initialize $\mathbf{W}$, I and A with small random numbers
while Not at convergence **do**
  Fix $\mathbf{W}$, I, A and update U (see equations 3.9 and 3.11)
  Fix $\mathbf{W}$, U, A and update I (see equations 3.9 and 3.11)
  Fix $\mathbf{W}$, U, I and update A (see equations 3.9 and 3.11)
  Compute $\mathbf{W} \doteq \mathbf{R} \times_1 U^* \times_2 I^* \times_3 A^*$
end while
Return full tensor $\widehat{\mathbf{R}} \doteq \mathbf{W} \times_1 U \times_2 I \times_3 A$

Algorithm 2 Direct-HOALS for CP factorization.

Input: dataset $\mathcal{D}$, desired rank K
Construct the initial (sparse) 3-order tensor $\mathbf{R}$ from $\mathcal{D}$
Initialize I and A with small random numbers
while Not at convergence **do**
  Fix I, A and update U (see equations 3.13 and 3.11)
  Fix U, A and update I (see equations 3.13 and 3.11)
  Fix U, I and update A (see equations 3.13 and 3.11)
end while
Return full tensor $\widehat{\mathbf{R}} \doteq [U, I, A]$

3.3 Higher Order Alternating Least Squares

HOALS is a tensor completion algorithm suitable for Tucker or CP decomposition. While the algorithm applies for any n-order tensor, we describe it for $n = 3$ for the sake of simplicity. The main idea of HOALS is that the decomposition of an n-order tensor can be parallelized over each of its modes independently, thus reducing it to a series of matrix decomposition operations. Furthermore, each matrix decomposition can be further parallelized across each component. The resulting process can be easily split over multiple machines, whose jobs are completely independent and whose results are merged only at the last step. Merging only at the last step also has the additional advantage of not having to synchronize the machine that is wait other machines to finish to continue.

A recommender system is provided with a tensor $\mathbf{R}$ which is only partially filled with the feedback in $\mathcal{D}$ (i.e., feedback for triples *(user, item, action)*) and the objective is to construct a full tensor $\widehat{\mathbf{R}}$ with entries that are as close as possible to the "true" ones (i.e., the feedback a user would give to an item for a specific action). Similar to the case of matrix completion, this problem is ill-posed (i.e., all tensors $\widehat{\mathbf{R}}$ with the same entries as $\mathbf{R}$ in $\mathcal{D}$ are equivalent) unless constraints on the structure of the tensor $\widehat{\mathbf{R}}$ are introduced. In particular, we introduce rank-constraints in the decomposition of $\widehat{\mathbf{R}}$ either following Tucker or CP.

Following the definition of CP decomposition, we try to complete $\mathbf{R}$ using tensors $\widehat{\mathbf{R}}$ that can be decomposed as

$$\widehat{\mathbf{R}}^{\text{CP}} \doteq [U, I, A] = \sum_{k=1}^{K} U_{:,k} \otimes I_{:,k} \otimes A_{:,k},$$

where $U \in \mathbb{R}^{n\times K}$, $I \in \mathbb{R}^{p\times K}$, $A \in \mathbb{R}^{q\times K}$ and K explicitly constraint the rank of the tensor and of the feature matrices representing respectively users, items, and actions. The entries of the approximated tensor are then

$$\widehat{\mathbf{R}}^{\text{CP}}_{uia} = \sum_{k=1}^{K} U_{u,k} \cdot I_{i,k} \cdot A_{a,k}.$$

Following the definition of Tucker decomposition, we work on tensors $\widehat{\mathbf{R}}$ that can be decomposed as

$$\widehat{\mathbf{R}}^{\text{Tucker}} \doteq \mathbf{W} \times_1 U \times_2 I \times_3 A,$$

where $U \in \mathbb{R}^{n\times K_U}$, $I \in \mathbb{R}^{p\times K_I}$, $A \in \mathbb{R}^{q\times K_A}$ are feature matrices for users, items, and actions, and $W \in \mathbb{R}^{K_U\times K_I\times K_A}$ is the core tensor, which encodes the interactions between the latent factors. In this case, the entries of the approximated tensor can be computed as

$$\widehat{\mathbf{R}}^{\text{Tucker}}_{uia} = \sum_{m=1}^{K_U} \sum_{l=1}^{K_I} \sum_{s=1}^{K_A} \mathbf{W}_{m,l,s} \cdot U_{u,m} \cdot I_{i,l} \cdot A_{a,s}.$$

Unlike CP decomposition, in this case we have the flexibility of choosing the ranks K_U, K_I, and K_A of each of the latent factors independently. This is particularly interesting in recommender systems where the set of users, items, and actions have different complexity and require different degrees of freedom in their description. For instance it is frequent to have million of users, several thousands of items and around 10 actions. Once a specific decomposition is chosen, the objective is to find the tensor $\widehat{\mathbf{R}}$ that minimizes the squared loss with a penalty term related to the latent factors weighted by the number of observed interactions. For instance, in the case of Tucker decomposition we want to find the components latent factors U, I, and A and the core tensor W that minimize the following loss function

$$\begin{aligned}\mathcal{L}(W, U, I, A) \quad &= \quad \sum_{u,i,a\in\mathcal{D}} \left(\mathbf{R}_{uia} - \widehat{\mathbf{R}}_{uia}\right)^2 \qquad (3.6)\\ &+ \quad \lambda \left(\sum_{u=1}^{n} n_u^{\mathcal{U}} ||U_u||^2 + \sum_{i=1}^{p} n_i^{\mathcal{I}} ||I_i||^2 + \sum_{a=1}^{q} n_a^{\mathcal{A}} ||A_a||^2 \right),\end{aligned}$$

Algorithm 3 Parallel-HOALS for Tucker factorization.

Input: dataset $\mathcal{D}$, desired ranks $\{k_d\}_{d=1}^3$
Construct the initial (sparse) 3-order tensor $\mathbf{R}$ from $\mathcal{D}$
Compute the three matrices unfolding $R_{(1)}$, $R_{(2)}$, $R_{(3)}$
In parallel apply [2]
  ALS-WR on $R_{(1)}$, $\widehat{R}_{(1)} = M^1 {V^1}^\top$ and set $U = M^1$
  ALS-WR on $R_{(2)}$, $\widehat{R}_{(2)} = M^2 {V^2}^\top$ and set $I = M^2$
  ALS-WR on $R_{(3)}$, $\widehat{R}_{(3)} = M^3 {V^3}^\top$ and set $A = M^3$
Compute $\mathbf{W} \doteq \mathbf{R} \times_1 U^* \times_2 I^* \times_3 A^*$
Return full tensor $\widehat{\mathbf{R}} \doteq \mathbf{W} \times_1 U \times_2 I \times_3 A$

Algorithm 4 Parallel-HOALS for CP factorization.

Input: dataset $\mathcal{D}$, desired rank K
Construct the initial (sparse) 3-order tensor $\mathbf{R}$ from $\mathcal{D}$
Compute the three matrices unfolding $R_{(1)}$, $R_{(2)}$, $R_{(3)}$
In parallel apply [2]
  ALS-WR on $R_{(1)}$, $\widehat{R}_{(1)} = M^1 {V^1}^\top$ and set $U = M^1$
  ALS-WR on $R_{(2)}$, $\widehat{R}_{(2)} = M^2 {V^2}^\top$ and set $I = M^2$
  ALS-WR on $R_{(3)}$, $\widehat{R}_{(3)} = M^3 {V^3}^\top$ and set $A = M^3$
Return full tensor $\widehat{\mathbf{R}} \doteq [U, I, A]$

where $n_u^{\mathcal{U}}$ (respectively $n_i^{\mathcal{I}}$, $n_a^{\mathcal{A}}$) is the number of known entries for user u (respectively item i, action a) in $\mathcal{D}$ and λ is a regularization parameter. As for ALS-WR this weighted penalisation allows to have stronger regularization to latent factors that have been observed a lot and thus are more likely to be overfitted.

Direct-HOALS. The loss function in Eq. 3.6 can be minimized with an alternating scheme similar to ALS-WR for matrix factorization (see Alg. 1). In this case, instead of repeating an alternate update on users and items only, also the action mode is updated while keeping the other two modes fixed. While a simple and smooth generalization of ALS-WR to tensor factorization, this algorithm, called *Direct*-HOALS, increases the complexity of the original ALS-WR by adding an update over the action mode. This process may be computationally expensive and prevent from scaling this algorithm to tensors of higher modes.

Parallel-HOALS. Instead of running ALS-WR over all modes iteratively, we design an alternative algorithm, called *Parallel*-HOALS, which runs ALS-WR separately on each unfolded tensor and collect its result to obtain the final tensor decomposition. The idea is to first construct the three unfolded tensors $R_{(1)}$, $R_{(2)}$, and $R_{(3)}$ and perform ALS-WR on each of them to find a suitable factorization of the corresponding matrix. The first factors of each of these decompositions can then be

[2]Remark that the previous main loop is now embedded in each of the ALS-WR computation, the equivalence is granted by Th. 1

collected to construct the factorization of the initial tensor. The resulting algorithm is illustrated in Alg. 3. Despite its difference with the sequential implementation of the ALS-WR idea of *Direct*-HOALS, the following theorem shows that (the proof is reported in a later section) *Parallel*-HOALS returns relevant matrices in all directions. Notice that since this result applies separately for each mode it can be extended to an arbitrary number of modes and it applies to both Tucker and CP decompositions. The biggest advantage of this version of HOALS is that the three applications of ALS-WR on $R_{(1)}$, $R_{(2)}$, and $R_{(3)}$ can be computed in parallel. Furthermore, since ALS-WR itself is highly parallelizable [Zhou et al., 2008], each instance of ALS-WR associated to any mode can be further parallelized, thus making HOALS very efficient without suffering any major overhead from the number of modes and elements in each mode. The update complexity for each event (i.e., update of the latent factor for one user, one item, or one action) for matrix $R_{(d)}$ is $\mathcal{O}\left({K_d}^2 n_s^d + {K_d}^3\right)$, where n_s^d is the number of non-zero elements on row $R_{(d),s}$ and K_d is the rank constraint. Since in HOALS we can parallelize the computation of each matrix and within each matrix, the complexity of one update is due to the largest latent dimension over the three dimensions, i.e., $\mathcal{O}\left(\max_{d\in\{1,2,3\},s}({K_d}^2 n_s^d + {K_d}^3)\right)$.

Additional advantages. Finally, we would like to emphasize some features that make the specific tensor decomposition in HOALS particularly suitable for recommender systems, unlike other tensor factorization methods. First, our method use Tikhonov regularization with different weights on each mode that regularized appropriately each dimension and is crucial for such application since the number of actions provided by each user can be highly heterogeneous. Second, tensor decompositions may appear difficult in practice because of memory issues that can arise from the additional modes. However thanks to its appropriate treatment of missing values and the low rank decomposition, memory is not an issue in practice since HOALS never requires to store the full tensor which is not the case with nuclear norm minimization approaches. Besides the full tensor never really need to be computed since the ratings of a user can be computed with the decomposition formula.

3.4 Algorithm guarantee

Theorem 1. *Solving U (resp. I, A) with alternating least square and k_U (resp. k_I, k_A) latent factor on the tensor $\widehat{\boldsymbol{R}}$ is equivalent to applying alternating least square on $R_{(1)}$ (resp. $R_{(2)}$, $R_{(3)}$) with k_U (resp. k_I, k_A) latent factor and identify U (resp. I, A) as the left side matrix in the matrix factorization.*

Remark For sake of clarity this theorem means that at any step of the update, updating U using ALS-WR on $R_{(1)}$ will give the same results as updating U using *Direct*-HOALS. However once the right matrix of $R_{(1)}$ decomposition is fully updated using ALS-WR (that is we update both matrices of the matrix factorization) we have no guarantee that the right matrix of the decomposition will contains the same values than the re-organisation of U, A and W after an update using *Direct-*HOALS. Nonetheless, this theorem shows that applying ALS-WR to compute the latent factors of a given dimension allows to improve their estimates provided that ALS-WR converges. Although interesting, we did not get the time to answer convergence question of HOALS but we feel like it should be related to the convergence of matrix factorization algorithms and more precisely of ALS-WR. See appendix A for an introduction to matrix completion convergence works.

Tucker Decomposition For sake of space, we use the following short summation notation

$$\sum_{m,l,s=1}^{k_U,k_I,k_A} \doteq \sum_{m=1}^{k_U}\sum_{l=1}^{k_I}\sum_{s=1}^{k_A}.$$

Besides, we recall that

$$\begin{aligned}\mathcal{L}(W,U,I,A) = &\sum_{u,i,a\in\mathcal{D}}\left(\mathbf{R}_{uia} - \widehat{\mathbf{R}}_{uia}\right)^2\\ &+\lambda\left(\sum_{u=1}^{n} n_u^{\mathcal{U}}||U_u||^2 + \sum_{i=1}^{p} n_i^{\mathcal{I}}||I_i||^2 + \sum_{a=1}^{q} n_a^{\mathcal{A}}||A_a||^2\right),\end{aligned}$$

Computing the partial derivative of $\mathcal{L}$ in Eq.3.6 w.r.t. U_{u_0,k_0}, we get:

$$\frac{1}{2}\frac{\partial \mathcal{L}}{\partial U_{u_0,k_0}} = \sum_{(i,a)\in\mathcal{I}_{u_0}}\Big(\sum_{m,l,s=1}^{k_U,k_I,k_A}\mathbf{W}_{m,l,s}U_{u_0,m}I_{i,l}A_{a,s} - \mathbf{R}_{u_0,i,a}\Big) \tag{3.7}$$

$$\times\sum_{l,s=1}^{k_I,k_A}\mathbf{W}_{k_0,l,s}I_{i,l}A_{a,s} + \lambda n_{u_0}^{\mathcal{U}}U_{u_0,k_0}, \tag{3.8}$$

where $\mathcal{I}_{u_0}$ denotes the items and actions seen by user u_0. Let us denote

$$\mathcal{V}_m^{i,a} = \sum_{l=1}^{k_I}\sum_{s=1}^{k_A}\mathbf{W}_{m,l,s}I_{i,l}A_{a,s} \tag{3.9}$$

and equalize the derivative above to 0, then we get the equation

$$\sum_{(i,a)\in\mathcal{I}_{u_0}}\Big(\sum_{m=1}^{k_U}U_{u_0,m}\mathcal{V}_m^{i,a}\mathcal{V}_{k_0}^{i,a}\Big) + \lambda n_{u_0}^{\mathcal{U}}U_{u_0,k_0} = \sum_{(i,a)\in\mathcal{I}_{u_0}}\mathcal{V}_{k_0}^{i,a}\mathbf{R}_{u_0,i,a} \tag{3.10}$$

We introduce the mapping g_1 function:

$$\begin{aligned} g_1 : [p]\times[q] &\rightarrow \{1,\cdots,pq\}\\ (i,a) &\mapsto \text{1-mode unfolding}\\ &\quad\ \text{corresponding column index}\end{aligned}$$

which maps item and action indices to a common index used to refer to the column of the 1-mode unfolding of the tensor. We can use this function to re-index $R^1 \in \mathbb{R}^{n,pq}$ such that $R^1_{u,g_1(i,a)} = r_{u,i,a}$ and $\mathcal{V}^1 \in \mathbb{R}^{pq,k_U}$ such that $\mathcal{V}^1_{g_1(i,a),m} = \mathcal{V}^{i,a}_m$. Let $\mathcal{G}_{u_0} = \{g : 1 \le g \le pq, R^1_{u_0,g} \neq 0\}$ we can then rewrite the left side (LS) of Eq. 3.10 and obtain $\forall u_0, \forall k_0$:

$$\begin{aligned}(LS) &= \sum_{m=1}^{k_U} \sum_{g \in \mathcal{G}_{u_0}} \mathcal{V}^1_{g,k_0} \mathcal{V}^1_{g,m} U_{u_0,m} + \lambda n^{\mathcal{U}}_{u_0} E_{k_0} U^\top_{u_0} \\ &= \sum_{m=1}^{k_U} [{\mathcal{V}^1}^\top_{\mathcal{G}_{u_0}} \mathcal{V}^1_{\mathcal{G}_{u_0}}]_{k_0,m} U_{u_0,m} + \lambda n^{\mathcal{U}}_{u_0} E_{k_0,:} U^\top_{u_0} \\ &= [{\mathcal{V}^1}^\top_{\mathcal{G}_{u_0}} \mathcal{V}^1_{\mathcal{G}_{u_0}}]_{k_0} U^\top_{u_0} + \lambda n^{\mathcal{V}}_{u_0} E_{k_0} U^\top_{u_0}\end{aligned}$$

Similarly for the right side

$$(RS) = \sum_{g \in \mathcal{G}_{u_0}} R^1_{u_0,g} \mathcal{V}^1_{g,k_0} = {\mathcal{V}^1_{\mathcal{G}_{u_0},k_0}}^\top {R^1_{u_0,\mathcal{G}_{u_0}}}^\top .$$

Thus $\forall u_0$, Eq. 3.10 becomes

$$\left({\mathcal{V}^1_{\mathcal{G}_{u_0}}}^\top \mathcal{V}^1_{\mathcal{G}_{u_0}} + \lambda n^{\mathcal{U}}_{u_0} E \right) U^\top_{u_0} = {\mathcal{V}^1_{\mathcal{G}_{u_0}}}^\top {R^1_{u_0,\mathcal{G}_{u_0}}}^\top$$

And finally,

$$U^\top_{u_0} = \left({\mathcal{V}^1_{\mathcal{G}_{u_0}}}^\top \mathcal{V}^1_{\mathcal{G}_{u_0}} + \lambda n^{\mathcal{U}}_{u_0} E \right)^{-1} {\mathcal{V}^1_{\mathcal{G}_{u_0}}}^\top {R^1_{u_0,\mathcal{G}_{u_0}}}^\top, \tag{3.11}$$

which is exactly the classic ALS-WR formula for matrix factorization applied to the 1-mode matrix unfolding (3.3), which concludes the proof. □

CP Decomposition Computing the partial derivative of $\mathcal{L}$ in Eq.3.6 w.r.t. U_{u_0,k_0}, we get:

$$\frac{1}{2} \frac{\partial \mathcal{L}}{\partial U_{u_0,k_0}} = \sum_{(i,a) \in \mathcal{I}_{u_0}} \left(\sum_{k=1}^{K} U_{u_0,k} I_{i,k} A_{a,k} - \mathbf{R}_{u_0,i,a} \right) \times I_{i,k} A_{a,k} + \lambda n^{\mathcal{U}}_{u_0} U_{u_0,k_0}, \tag{3.12}$$

where $\mathcal{I}_{u_0}$ denotes the items and actions seen by user u_0. As previously, let us denote

$$\mathcal{V}^{i,a}_k = I_{i,k} A_{a,k} \tag{3.13}$$

and set the derivative above to 0, then we get the equation

$$\sum_{(i,a) \in \mathcal{I}_{u_0}} \left(\sum_{k=1}^{K} U_{u_0,k} \mathcal{V}^{i,a}_k \mathcal{V}^{i,a}_{k_0} \right) + \lambda n^{\mathcal{U}}_{u_0} U_{u_0,k_0} = \sum_{(i,a) \in \mathcal{I}_{u_0}} \mathcal{V}^{i,a}_{k_0} \mathbf{R}_{u_0,i,a}$$

We use again the function g_1 to re-index $R^1 \in \mathbb{R}^{n,pq}$ such that $R^1_{u,g_1(i,a)} = r_{u,i,a}$ and $\mathcal{V}^1 \in \mathbb{R}^{pq,K}$ such that $\mathcal{V}^1_{g_1(i,a),k} = \mathcal{V}^{i,a}_k$. Let $\mathcal{G}_{u_0} = \{g : 1 \leq g \leq pq, R^1_{u_0,g} \neq 0\}$, we rewrite the left side (LS) of Eq. 3.14 and obtain $\forall u_0, \forall k_0$

$$\begin{aligned}(LS) &= \sum_{k=1}^{K} \sum_{g \in \mathcal{G}_{u_0}} \mathcal{V}^1_{g,k_0} \mathcal{V}^1_{g,k} U_{u_0,k} + \lambda n^{\mathcal{U}}_{u_0} E_{k_0} U^\top_{u_0} \\ &= \sum_{k=1}^{K} [\mathcal{V}^1{}^\top_{\mathcal{G}_{u_0}} \mathcal{V}^1_{\mathcal{G}_{u_0}}]_{k_0,k} U_{u_0,k} + \lambda n^{\mathcal{U}}_{u_0} E_{k_0,:} U^\top_{u_0} \\ &= [\mathcal{V}^1{}^\top_{\mathcal{G}_{u_0}} \mathcal{V}^1_{\mathcal{G}_{u_0}}]_{k_0} U^\top_{u_0} + \lambda n^{\mathcal{V}}_{u_0} E_{k_0} U^\top_{u_0}\end{aligned}$$

Similarly for the right side

$$(RS) = \sum_{g \in \mathcal{G}_{u_0}} R^1_{u_0,g} \mathcal{V}^1_{g,k_0} = \mathcal{V}^1_{\mathcal{G}_{u_0},k_0}{}^\top R^1_{u_0,\mathcal{G}_{u_0}}{}^\top.$$

We can then conclude as previously. □

3.5 Experiments

We tested both the accuracy and the scalability of our method on different types of datasets:

- rather small datasets for recommendation problem but still large for common machine learning problem. These datasets allowed us to make a lot of experiments on a substantial number of algorithms and tune them quite effectively. These datasets also emphasize the benefit of using tensor decomposition instead of matrix decomposition.
- a bigger dataset to test the scalability performance as a function of the dataset size and the number of machines used.

3.5.1 Testing accuracy

We report an empirical comparison of a number of matrix and tensor decomposition methods. The objective is two-fold: **1)** investigate whether tensor completion enables an effective transfer across different actions that improves over matrix completion where each action is treated separately, **2)** compare HOALS to state-of-the-art methods.

Dataset. We use two real-world datasets:

- **E-commerce dataset** [Warlop, 2016]: this dataset contains historical interactions (click, add to cart, purchase) from a real e-commerce website of 1290 users and 390 items, resulting in around 120k *(user, item, action, value)* quadruplets. The *value* is the number of times the user performed the action on the item.

- **Last.fm Dataset - 1K users** [Celma, 2009]: it is formed of around 20M lines representing the track of an artist that a user is listening to at a given timestamp. We transformed the dataset in order to get the quadruplets *(user, artist, moment, value)*, where the *moment* represents the time of day clustered into three periods: *morning* from 6 a.m. to 2 p.m., *afternoon* from 2 p.m. to 10 p.m., and *night* from 10 p.m. to 6 a.m. The value counts the number of times the user listened to that artist in that period. The final dataset is composed of 510k quadruplets.

Algorithms. Beside HOALS, we tested four other state-of-the-art algorithms, two with different initialization

- **MI-SVD**: multiple independent SVD applied to each slice of the tensor,
- **HOSVD v0/vMean**: as proposed by [Symeonidis et al., 2008], filling missing values with either 0 or the mean of the modes,
- **MI-ALS-WR**: the classic ALS-WR applied to each slice of the tensor, that is three independent ALS-WR,
- **CP-ALS-R v0/vMean** (aka PARAFAC-ALS) as in Alg.1 of [Hu et al., 2013], filling missing values with 0 or the mean of the modes. The algorithm is detailed in section 3.6.2. We used the *cp_als* python code from the sktensor package,
- **tucker/CP-HOALS**, our parallel algorithm described in Alg. 3. The implementation is available at [Warlop, 2016].
- **parafac-HOALS**, our model. The code of Alg. 4 is available at [Warlop, 2016].

The *vMean* versions had only been tested on the E-commerce dataset because filling with non zero values require much more memory, since sparse representations of the structures cannot be used any more. As introduced in section 3.2, a more suitable version of ALS-WR for implicit feedback like the Last.fm dataset is the algorithm proposed by [Hu et al., 2008] called implicit ALS-WR. In the following, we use exactly the same approach to obtain an implicit version of HOALS. Experiments test both classic and implicit methods for ALS-WR and HOALS, which are referred to *MI*-ALS-WR and HOALS for the classic version and *imp-MI*-ALS-WR and *imp*-HOALS for the implicit counterparts. Finally, we also tried the algorithms proposed by [Romera-Paredes and Pontil, 2013] and [Acar et al., 2010] but their solutions could not scale to the size of our datasets.

Experimental protocol. We first executed a preprocessing step.

1. We filtered out items seen by less than 5 users,

2. We define a maximum threshold for each action used to cap all the values (i.e., $r_{uai} = \min\{r_{uai}, threshold_a\}$). This prevents from having outliers that tend to degrade the performance. The click threshold was set to 50, the ATC to 20 and buy to 10.

3. We scaled all values between 0 and 10 in order to have comparable slices. This is needed since click, ATC, and buy actions are at very different scales.

For this experiment we performed 5-cross validation using the stratified cross validation introduce in 2.2.1 with 80% of users in the training sets and 20% of users in the test set and 70% of their ratings added to the training set. For evaluating the recommendation performance, we use the rank-measure (2.5) error, noted $\overline{\rho}$, introduced by [Hu et al., 2008], which is more suitable for implicit feedback datasets.
Results. The results are reported in Table 3.1. For Tucker models we tried 10, 50, 100 and 200 number of latent vectors for each possible tuning combination for the user and item slice, and 2 latent vectors for the action slice, since the depth of the tensor is always 3 in our cases. For CP decompositions, given that the rank has to be the same for all modes, we tried $2, 3, 4$, 5 and 10 latent vectors for all modes. We were not able to test higher number of latent vectors because of memory issue when trying to factorize the third unfolding matrix of size $(q \times np)$, which illustrates one of the drawbacks of CP decompositions. Furthermore, notice that increasing the number of latent vectors for the CP decomposition is not really meaningful since the action mode only contains three different actions. Finally for matrix decomposition we tried 10, 50, 100 and 200 number of latent vectors for each slice. For the sake of space we do not provide the parameters tuning results for each combination (several hundreds) and only report the performance for each model in their best configuration for each action.
E-commerce dataset. In e-commerce applications, recommendations are usually performed using only the click action because of the sparsity of the buy action. For instance, in this dataset the sparsity level for click, ATC, and buy are respectively 85%, 94% and 98%. However, the results in Table 3.1 show that Tucker HOALS (implicit or not) outperformed every other model obtaining a $\overline{\rho}$ measure around 28% for the implicit version, even for the buy action, while other models are around 35%. The implicit Tucker HOALS improves the classic Tucker HOALS performance by more than two points, which illustrates the benefit of such a modelisation. Even the CP models outperformed the other factorization methods with a $\overline{\rho}$ measure around 32%. This indicates that the HOALS is able to perform better recommendation for the optimization of the purchase and is more suitable than the other tensor factorization methods thanks to its ability to deal with missing values. Furthermore, tensor factorization methods achieve better results than their matrix factorization counterparts (HOALS vs ALS-WR and HOSVD vs SVD), which shows the advantage of transferring information across actions within this dataset. However, the HOSVD improvement over MI-SVD is relatively small, about 1 point, whereas the tucker

HOALS improvement over MI-ALS-WR is around 5 points for the classic algorithm and 12 points for the implicit one. These results show the effectiveness of HOALS in overcoming the sparsity issue by sharing knowledge across the different actions. Finally, we see no real improvement in the initialisation of missing values with 0 or with mean values.

Last.fm dataset - 1K users. For the Last.fm dataset (Table 3.2), tucker HOALS (implicit or not) obtains the best results with a $\overline{\rho}$ measure around 25%, two points better than HOSVD which is the next best model. Surprisingly, it is not the implicit version that achieve the best performance but the classic one, although both methods have really close performance (less than 0.6 point). Our CP method achieves relatively bad performance with a $\overline{\rho}$ measure around 32%. We think this can come from three reasons. The first one is due to the constraint of CP decomposition that imposes to have the same number of latent factors for each mode, which is a strong drawback for very imbalance tensor's dimensions, as it is the case here. The second one is the low number of features for the users and items modes that we could not increase because of computational issues. Both these explanations are common to CP-ALS-R and CP-HOALS. Finally, given the good performance of the HOSVD decomposition, we conjecture that treating missing values as 0 as done by the SVD composition models is not completely unreasonable in this case, since a user never listens to an artist if he/she never heard about or do not like her. The combination of the two precedents reasons and this observation explain the bad performance of CP-HOALS on this dataset that has difficulty to learn good user and item representation in a very low dimension with only the known entries. However, once again the good performance of the tensor factorization models prove the effectiveness of transfer learning in this context. Indeed the gain between the best ALS-WR and the best tucker HOALS is around 7 points.

Other results. Besides those algorithms we also tried other techniques and algorithm that did not lead to satisfying results. For our HOALS implementation, we tried to model the movie recommendation problem as a tensor completion problem where each slice would represent a rating value and implement this technique on the movielens-1M dataset. To clarify things, if a movie can be rated between 1 and 5, we would have a tensor with five slices, where entries $r_{u,i,a} = 1$ is user u rated movie i with a rating of a, and 0 otherwise. To infer the final user-item rating we first tried to select the maximum value in the rating fiber, thus the predicted rates are integers between 1 and 5 while classic matrix factorization outputs real values. This give a worse RMSE than classic ALS-WR on matrix completion. We assume that this come from the fact that ratings are normally distributed around 3.5 thus when ALS-WR predict 3.5 it is often a good prediction whether the actual rating is 3 or 4 with a squared error of 0.25 whereas with our technique if we predict 4 instead of 3 the squared error contribution is 1. To overcome this problem we computed the weighted average on the fiber. For instance if the predicted value of user u on movie i for each of the five slices are respectively $0.15, 0.2, 0.5, 0.3, 0.2$ (it does not sum to 1), the predicted rating is $r_{u,i} \doteq \frac{0.15\times1+0.2\times2+0.5\times3+0.3\times4+0.2\times5}{0.15+0.2+0.5+0.3+0.2} \approx 3.15$. This technique gave similar RMSE to classic ALS-WR on matrix completion but no real improvement. Another technique we studied for HOALS is to use contextual information as additional slice, where each slice represent one contextual. For instance if we know that movie i is a thriller, then $r_{u,i,a} = 1$ for all users if slice a correspond to the thriller context. Unfortunately this modelisation consumed too much memory for the computer and did not run even on a small dataset like movielens-100k. In addition to HOALS, we tried the algorithms proposed by [Romera-Paredes and Pontil, 2013] and [Acar et al., 2010] but their solutions could not scale to our large datasets.

3.5.2 Testing scalability

For this experiment we used a real world dataset referred as **big E-commerce dataset** [Warlop, 2016]. The full dataset is composed of 100k users, around 25k items and 4 actions (view the product, click on the product, add it to cart or buy it) resulting in around 53M (user, item, action, value) quadruplets. The view action is a passive action, a user is exposed to an item by the website but cannot directly decide to see it or not. Thus this action is not really interesting to predict but can be very informative. Indeed a user may be more likely to click, add to her cart or buy a product if she sees it a lot. It is then a valuable context information that can be easily taken into account using tensor decomposition.

To test the scalability we measured the execution time when varying the size of the dataset and number of machines. For the size of the dataset we selected either $1k$, $2k$ or $3k$ users and all the items (since the files are split every 1k users), knowing that $1k$ users result in a dataset of around 9Mo and 530k lines, i.e., quadruplets. We performed our experiments on *Google Cloud* on the n1-standard-2 machine, which is standard 2-CPU machine with 2 virtual CPUs and 7.5 GB of memory. When running the three decompositions in three independent cluster, the execution time depends on the execution time of the longest decomposition. We then report here the execution time of the longest decomposition for each dataset size.

Results. Fig. 3.5 compares *Direct* and *Parallel*-HOALS. We see that the execution times decreases linearly as the number of workers increases. Doubling the number of workers does not divide the execution time by a factor of 2 (for instance 2 workers, 1k users need 1200 seconds and with 4 workers it need 780 seconds) because of the set up and the communication time. Adding 1CPU decreases the running time in average of 200 (resp. 300 and 400) seconds for 1k users (resp. 2k and 3k). Thus adding users affect mainly the set up and communication time but the parallelisation is more and more effective. We thus see that the HOALS algorithm scale easily for big dataset thanks to its high parallelisation power. Furthermore, this improvement would be greatly amplified as the number of modes increases. For instance, the performance of RS in e-commerce can be significantly improved by adding new contexts such as the day of the week and the season of the year, thus resulting in a 5-order tensor completion problem.

model	dimension 1	dimension 2	action	$\overline{\rho_a}$
imp Tucker-HOALS	10	200	click	**28.79**
Tucker-HOALS	200	200	click	30.34
CP-HOALS	5	-	click	31.92
imp CP-HOALS	2	-	click	32.01
CP-ALS-R	2	-	click	32.20
MI-ALS-WR	50	-	click	35.89
HOSVD v0	10	200	click	36.39
HOSVD vMean	10	100	click	36.42
imp MI-ALS	100	-	click	36.66
MI-SVD	10	-	click	37.11
imp Tucker-HOALS	10	200	atc	**28.58**
Tucker-HOALS	200	200	atc	31.63
imp CP-HOALS	2	-	atc	31.70
CP-HOALS	5	-	atc	31.91
CP-ALS-R	2	-	atc	32.11
HOSVD vMean	10	200	atc	34.94
HOSVD v0	10	200	atc	35.08
MI-SVD	10	-	atc	36.25
MI-ALS-WR	10	-	atc	36.63
imp MI-ALS	100	-	atc	43.14
imp Tucker-HOALS	10	200	buy	**28.13**
CP-ALS-R	4	-	buy	31.19
imp CP-HOALS	2	-	buy	31.45
CP-HOALS	5	-	buy	31.74
Tucker-HOALS	10	200	buy	32.38
HOSVD vMean	10	100	buy	34.56
HOSVD v0	10	50	buy	34.88
MI-SVD	10	-	buy	35.49
MI-ALS	10	-	buy	36.79
imp MI-ALS-WR	50	-	buy	41.96

Table 3.1: $\overline{\rho_a}$ measure (rank-measure) on the E-commerce dataset ordered by decreasing performance. For matrix factorization and CP tensor factorization, *dim*1 refers to the common latent space. For tensor factorization, *dimension* 1 and *dimension* 2 refers to the number of latent vectors for the user and item space respectively and *dimension* 3 is always equal to 2.

3.6 Related Works

As introduce in subsection 2.1.1 there are three families of algorithms to predict user-item affinities: content-based, collaborative filtering and hybrid methods. This work is more related to collaborative filtering since we do not assume the existence

model	dimension 1	dimension 2	moment	$\overline{\rho_a}$
Tucker-HOALS	200	200	morning	**23.65**
HOSVD	200	200	morning	24.58
imp Tucker-HOALS	200	200	morning	26.54
imp MI-ALS-WR	50	-	morning	31.62
imp Tucker-HOALS	10	10	morning	30.40
CP-ALS-R	10	-	morning	31.76
imp CP-HOALS	10	-	morning	34.42
CP-HOALS	4	-	morning	36.98
MI-SVD	50	-	morning	34.72
MI-ALS	10	-	morning	40.53
Tucker-HOALS	200	200	afternoon	**23.62**
imp tucker-HOALS	200	200	afternoon	25.61
HOSVD	100	100	afternoon	27.26
imp Tucker-HOALS	10	10	afternoon	29.47
imp MI-ALS-WR	50	-	afternoon	30.14
CP-ALS-R	10	-	afternoon	30.85
imp CP-HOALS	4	-	afternoon	32.57
CP-HOALS	5	-	afternoon	37.25
MI-SVD	50		afternoon	34.00
MI-ALS	10	-	afternoon	39.25
Tucker-HOALS	200	200	night	**24.81**
imp Tucker-HOALS	200	200	night	25.28
HOSVD	100	100	night	27.61
imp Tucker-HOALS	10	10	night	29.97
imp MI-ALS-WR¿	50	-	night	31.49
CP-ALS-R	10	-	night	31.85
imp CP-HOALS	4	-	night	32.92
CP-HOALS	3	-	night	37.05
MI-SVD	10	-	night	35.17
MI-ALS	10	-	night	40.30

Table 3.2: $\overline{\rho_a}$ measure (rank-measure) on the Last.fm dataset ordered by decreasing performance. For matrix factorization and CP tensor factorization, *dimension*1 refers to the common latent space. For tensor factorization, *dimension* 1 and *dimension* 2 refers to the number of latent vectors for the user and item space respectively and *dimension* 3 is always equal to 2. Due to memory limitations, we were not able to perform a CP-ALS-R in dimension greater than 10, we thus compare to HOALS in dimension 10 as well.

of contextual information but the latent factors obtained with HOALS could be used in a hybrid methods if combine with other algorithms.

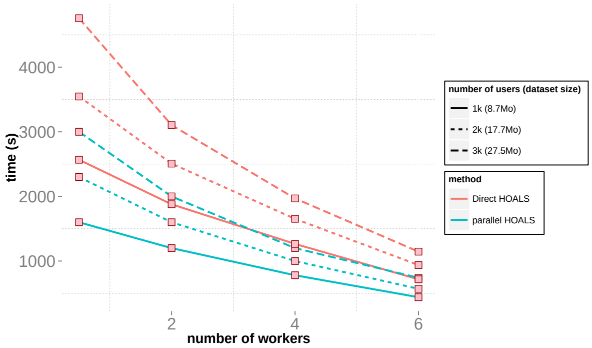


Figure 3.5: Result of the scalability test of parallel HOALS ran with the implicit version with rank constraints of 200 for users and items and 2 for actions. Each workers has 2CPUs.

3.6.1 Collaborative Filtering

Early methods

At the beginning of recommender systems, algorithms explicitly rely on the idea that users that agreed in the past will agreed in the future. All the problem was then to find for each user the set of users with whom she agreed in the past, that is find similar user. In machine learning this problem consist in finding the neighbourhood of each user and leads to number of **neighbourhood-based Collaborative Filtering methods**. For instance [Resnick et al., 1994] used the Pearson correlation measure to compute the similarity $w_{a,u}$ between the active user a and the user u based on the set of commonly rated items $I_{a,u}$:

$$w_{a,u} = \frac{\sum_{i \in I_{a,u}} (r_{a,i} - \overline{r_a})(r_{u,i} - \overline{r_u})}{\sum_{i \in I_{a,u}} (r_{a,i} - \overline{r_a})^2 \sum_{i \in I_{a,u}} (r_{u,i} - \overline{r_u})^2} \tag{3.14}$$

where $\overline{r_x}$ is the average rating given by user x. Then to compute the rating prediction of item i for the active user a, $\hat{r}_{a,i}$, the K most similar user according to $w_{a,u}$ that have rated item i, denoted $N_K(a)$, are mixed to compute the following weighted prediction:

$$\hat{r}_{a,i} = \overline{r_a} + \frac{\sum_{u \in N_K(a)} w_{a,u}(r_{u,i} - \overline{r_u})}{\sum_{u \in N_K(a)} w_{a,u}} \tag{3.15}$$

Many other similarity measures have been investigated such as the cosine angle between users, the Spearman rank correlation, Kendall's τ correlation, mean squared differences, entropy, adjusted cosine similarity, ... ([Herlocker et al., 1999b], [Su and Khoshgoftaar, 2009]). Along the years, such techniques have been improved to take into account more precise information. **Significance Weighting** ([Herlocker et al., 1999b]) and **Default Voting** ([Breese et al., 1998]) have been used to reduce the bias of computing correlation based on very few co-rated items. **Inverse User Frequency** ([Breese et al., 1998]) to give a specific weight on each item based on the user that have rated this item. Finally **Case Amplification** ([Breese et al., 1998]) exponentially multiply the weight to give higher importance to highly similar users. All those techniques are also called User-User collaborative filtering because the rating prediction is based on user-user similarity measure. One can also perform Item-Item collaborative filtering (see [Linden et al., 2003]) by reversing the formula to compute item-item similarity and compute similar weighted mean using item in the neighbourhood of the active item previously rated by the active user to compute the desired user-item rating. The choice of User-User collaborative filtering or Item-Item collaborative filtering will depend on the available data. If there are much more users than items, item-item similarities will be more precise than user-user similarities because they will be more likely to have a denser vector.

In general such techniques really suffer from scaling issues when the number of users or items is high, which is almost always the case in recommender systems. Indeed, to predict just one rating for a given user, one must find the subset of user that have rated the same item, select the top K similar user and then compute the weighted mean. This computation make the solution infeasible in practice. Besides, this solution has the drawback to rely a lot on human definition of similarity and importance measure. Finally, those methods can also suffer from blind spots as some items never intersect with the user neighbourhood in case of User-User collaborative filtering. Current methods on the other hand focus on scalability, data defined users and items descriptors and global prediction. They will be the focus of the following section.

State-of-the-art

To solve those issues, many methods rely on variations of the common idea of low rank matrix and thus perform matrix factorization. Here we present most common algorithms, except ALS-WR that has already been introduced.

1- APG [Toh and Yun, 2010] - Accelerated Proximal Gradient algorithm for matrix completion problems solve a nuclear norm regularized least square problem in order to obtain the full matrix. Ideally, the CF problem should be to find the matrix X with the smallest possible rank that fit the known values:

$$\min_{X \in \mathbb{R}^{n \times p}} \{rank(X); X_{u,i} = R_{u,i},\ \forall (u,i) \in \mathcal{D}\} \tag{3.16}$$

which is an NP-hard nonconvex optimization problem. However, a convex relaxation of this problem is to minimize the nuclear norm (sum of the singular values of the matrix), [Fazel, 2002], which is the best convex approximation of the rank function over the unit ball matrices, over the same constraints:

$$\min_{X \in \mathbb{R}^{n \times p}} \{||X||_*; X_{u,i} = R_{u,i}, \ \forall (u,i) \in \mathcal{D}\} \tag{3.17}$$

More generally, APG algorithm optimize the unconstrained nonsmooth convex minimization problem of the form:

$$\min_{X \in \mathbb{R}^{n \times p}} F(X) \doteq f(X) + P(X) \tag{3.18}$$

where $P : \mathbb{R}^{n \times p} \rightarrow (-\infty, \infty]$ is a proper, convex, lower semicontinuous function and f is convex smooth. Thus for matrix completion problem we have $P(X) = \mu||X||_*$ and $F(X) = \sum_{u,i \in \mathcal{D}} ||R_{u,i} - X_{u,i}||_2$. The solution is obtained by iteratively using the singular value thresholding algorithm (see [Cai et al., 2010]) on a quadratic approximation of F.

2- FM [Rendle, 2010] - Factorization Machines is a generalization of matrix factorization. Here the entries are not arrange in a matrix but in a array $X \in \mathbb{R}^{m \times q}$ whose lines contains the context and a vector $Y \in \mathbb{R}^m$ that stock the feedback as in classic supervised learning. For instance the first n values of observation X_l will encode the user id as a binary vector (that is $X_l[1:n] = 0$ except at position u for user u), the next p values will encode the item id as a binary vector (p zeros except at position $n+i$ for item i), the following value can for instance represent the age of the user, the next one the category of the item, etc., and Y_l is the feedback of user u on item i. In FM the function that link observation to output has the following form:

$$\hat{y}(x) = w_0 + \sum_{j=1}^{q} w_j x_j + \sum_{j=1}^{q} \sum_{s=j+1}^{q} < V_j, V_s > x_j x_s \tag{3.19}$$

Where $< V_j, V_s >= \sum_{k=1}^{K} V_{j,k} V_{s,k}$ In particular, if $q = n + p$ and thus the context is restricted to user and item ids, then x is not zero only at position u for user u and $n+i$ for item i, and then $\hat{y}(x) = w_0 + w_u + w_{n+i} + V_u V_{n+i}^T$ which falls back to classic matrix factorization with bias terms. Parameters estimation is then done using a fast SGD that use a optimization rewritting of the equation of the following objective function for each parameter:

$$\min_{\theta \in \Theta} \sum_{x,y \in S} \left(\hat{y}(x) - y\right)^2 + \sum_{\theta \in \Theta} \lambda_\theta \theta^2 \tag{3.20}$$

where λ_θ is a regularization (hyper-)parameter for the model parameter θ. To summarize, FM is a simple way to express pairwise interaction in sparse dataset and thus very suitable for recommender systems application. Even though the equation is written for 2-way FM, the formula can be extended to dimension d as follow:

$$\hat{y}(x) = w_0 + \sum_{j=1}^{q} w_j x_j + \sum_{l=2}^{d} \sum_{i_1=1}^{q} \cdots \sum_{i_l=i_{l-1}+1}^{q} \left(\prod_{j=1}^{l} x_{i_j} \right) \left(\sum_{f=1}^{k_l} \prod_{j=1}^{l} v_{i_j,f}^{(l)} \right) \quad (3.21)$$

3- LLORMA [Lee et al., 2016] - Local Low-Rank Matrix Approximation extend the concept of low rank matrix approximation using ensemble methods. LLORMA algorithm learn q independent weighted LRMA where the weight depend on the distance between each user and a user anchor and each item and an item anchor. For instance the distance can be computed using the angle between the latent factors that have been previously computed using one LRMA. The user (resp. item) anchor is chosen at random among all users (resp. items). The distance is then pass through a kernel (for instance the Epanechnikov kernel):

$$\left[K_{user}^{(t)}\right]_u = (1 - d(anchor^{(t)}, u)^2) 1_{\{d(anchor^{(t)},u)<h\}} \quad (3.22)$$

$$\left[K_{item}^{(t)}\right]_i = (1 - d(anchor^{(t)}, i)^2) 1_{\{d(anchor^{(t)},i)<h\}} \quad (3.23)$$

That is if the user (resp. item) is far from the current anchor, the weight will be null and close to 1 otherwise. Thus, at a given step, only part of the users and items will be selected to form the input. The objective function solved by LLORMA for the t^{th} LRMA is then:

$$\min_{U \in \mathbb{R}^{n \times K}, V \in \mathbb{R}^{n \times K}} [K_{user}^{(t)}]_u [K_{item}^{(t)}]_i \sum_{u,i \in \mathcal{D}} \left(r_{ui} - U_u V_i^T \right)^2 + \lambda_U ||U_u||^2 + \lambda_V ||V_i||^2 \quad (3.24)$$

The idea is that, with a sufficiently large value of q, the model will learn different factorization of subset of the initial matrix, where each subset is more likely to be low rank than the full initial matrix. Eventually, the prediction is obtained by computing the weighted mean as follow:

$$\hat{R}_{ui} = \sum_{t=1}^{q} \frac{\left[K_{user}^{(t)}\right]_u \left[K_{item}^{(t)}\right]_i}{\sum_{j=1}^{q} \left[K_{user}^{(j)}\right]_u \left[K_{item}^{(j)}\right]_i} \hat{U}_u^{(t)} \hat{V}_i^{(t)} \quad (3.25)$$

This algorithm showed very good performances but this come with a much higher computational cost.

4- Logistic MF [Johnson, 2014] - Logistic Matrix Factorization focus on implicit feedback and adopt a probabilistic point of view of matrix factorization. [Johnson, 2014] is at [Hu et al., 2008] what logistic regression is at linear regression. Let l_{ui} be the event that the user u interacted with item i. This event is then model has a logistic function of the inner product of user and item latent factors plus an user and item biases (respectively β_u, and β_i):

$$\mathbb{P}(l_{ui}|U_u, V_i, \beta_u, \beta_i) = \frac{\exp(U_u V_i^T + \beta_u + \beta_i)}{1 + \exp(U_u V_i^T + \beta_u + \beta_i)} \tag{3.26}$$

The user and item biases allows to capture user engagement and item popularity. Besides, as [Hu et al., 2008], the authors introduced a confidence weight on the entries proportional to the true value r_{ui}, for instance $c_{ui} = \alpha r_{ui}$. Then each non-zero entries serve as c_{ui} positive entries, and each zero entries as a single negative one. Further assuming that all entries of R are independent lead to the following likelihood:

$$\mathcal{L}(R|U, V, \beta) = \prod_{u,i} \mathbb{P}(l_{ui}|U_u, V_i, \beta_u, \beta_i)^{\alpha r_{ui}} (1 - \mathbb{P}(l_{ui}|U_u, V_i, \beta_u, \beta_i)) \tag{3.27}$$

Placing zero-mean spherical Gaussian priors on user and item latent factors as:

$$\mathbb{P}(U, \sigma^2) = \prod_u \mathcal{N}(U_u|0, \sigma_u^2 I), \quad \mathbb{P}(V, \sigma^2) = \prod_i \mathcal{N}(V_i|0, \sigma_i^2 I), \tag{3.28}$$

lead to the following log-likelihood:

$$\log \mathbb{P}(R|U, V, \beta) = \sum_{u,i} \alpha r_{ui} (U_u V_i^T + \beta_u + \beta_i) \tag{3.29}$$

$$- (1 + \alpha r_{ui}) \log(1 + \exp(U_u V_i^T + \beta_u + \beta_i)) \tag{3.30}$$

$$- \frac{\lambda}{2} (||U_u||^2 + ||V_i||^2) \tag{3.31}$$

where constant term are replaced by the scaling parameter λ. We see that the zero mean Gaussian prior is equivalent to ℓ_2-regularization. The final objective is then to maximize the log posterior of 3.29:

$$\arg\max_{U,V,\beta} \log \mathbb{P}(R|U, V, \beta) \tag{3.32}$$

The parameters can then be learned by alternating gradient descent, and optimized by choosing an adaptive gradient step size via AdaGrad. Note that in this formulation missing values are actually treated as 0 that is negative feedback. Besides introducing bias this assumption can lead to infeasible computation and memory issues. To limit those phenomenon, the authors proposed to only sample some missing values to be considered as 0 entries.

5- RSVD [Paterek, 2007] - Regularized Singular Value Decomposition solve the sparse matrix factorization problem using stochastic gradient descent on the regularized sum of squared residuals:

$$\min_{U\in\mathbb{R}^{n\times K}, V\in\mathbb{R}^{n\times K}} \sum_{u,i\in\mathcal{D}} \left(r_{ui} - U_u V_i^T\right)^2 + \lambda\left(||U_u||^2 + ||V_i||^2\right) \tag{3.33}$$

Thus, unlike ALS-WR that has a closed form solution, RSVD update the user and item latent factors with SGD one feature at a time. Thus the update equation rule for one feature of user u is $U_{uk} = U_{uk} + lrate\left((r_{ui} - \hat{r}_{ui})U_{uk} - \lambda U_{uk}\right)$, where $lrate$ is the learning rate, usually 0.001. One can find the formula for V by symmetry of the problem. Adding user ($b_u \in \mathbb{R}$), item ($b_i \in \mathbb{R}$) and global bias to the formula ($\hat{r}_{ui} = U_u V_i^T + b_u + b_i + \mu$) lead to the Improved RSVD model also referred as RSVD2. The global bias can be seen as the average rating of the database, the user bias measure the tendency of the user to give higher or lower feedback values than other users, the item bias the tendency of an item to receive a higher or lower feedback than other, which then leave the model to measure separately the join user-item interaction.

6- SVD++ [Koren, 2008] extend the latent factor model of [Paterek, 2007] by incorporating implicit feedback. In this context an implicit feedback is an information of an interaction of the user with the system without explicitly rating the items. For instance we can know that the user has viewed some movies but we do not know the ratings, or click on a product but we do not know if she bought it or not. Let N_u be the set of item that user u interacted with, $I_u \subset N_u$, and $Y_i \in \mathbb{R}^K$ the latent factor of item i for the implicit feedback. Then the desired output is defined as

$$\hat{r}_{ui} = \mu + b_u + b_i + V_i^T\left(U_u + |N_u|^{-\frac{1}{2}} \sum_{j\in N_u} Y_j\right) \tag{3.34}$$

where μ is the global bias, b_u and b_i the user and item bias and $|N_u|$ the cardinality of N_u. The parameters are then learn by SGD on the residuals squared errors:

$$\begin{aligned} \min_{\mu,b,U,V,Y} \sum_{u,i\in\mathcal{D}} (r_{ui} - \hat{r}_{ui})^2 \qquad\qquad\qquad\qquad (3.35)\\ + \lambda\left(b_u^2 + b_i^2 + \sum_{u\in I_i} ||U_u||^2 + \sum_{i\in I_u} ||V_i||^2 + \sum_{j\in N_u} ||Y_j||^2\right) \end{aligned}$$

3.6.2 Tensor Factorization

PARAFAC decomposition without missing data.

1- CP-ALS-R In [Hu et al., 2013], the authors focus on the problem on cross domain recommendation, for instance books, musics and movies simultaneously for the same users. Instead of concatenating the domain in order to create a very wide matrix, they look for a tensor formulation of the problem where each slice of a tensor capture a domain that will be filled by CP decomposition. The advantage is that each domain will have its own latent factors that will not have to be shared with other domains. The problem of finding a CP decomposition of a generic tensor $\mathbf{X} \in \mathbb{R}^{n\times p\times q}$ is restated as a least-squares problem, i.e., finding the factors $A \in \mathbb{R}^{n\times K}$, $B \in \mathbb{R}^{p\times K}$, $C \in \mathbb{R}^{q\times K}$ that minimize the loss

$$f(A, B, C) = \frac{1}{2}||\mathbf{X} - [A, B, C]||_F^2 + \frac{\lambda_A}{2}||A||_F^2 + \frac{\lambda_B}{2}||B||_F^2 + \frac{\lambda_C}{2}||C||_F^2. \quad (3.36)$$

Computing the derivative and setting it to zero yields to a closed-form expression for each matrix, based on the pseudo-inverse property of Khatri-Rao product. For instance, when fixing matrices B and C, the derivative with respect to A is:

$$\frac{\partial f}{\partial A} = (X_{(1)} - Y_{(1)})(C \odot B) + \lambda_A A \quad (3.37)$$

where $\mathbf{Y} = [A, B, C]$ and then

$$A = X_{(1)}(C \odot B)(B^T B \otimes C^T C + \lambda_A I)^* \quad (3.38)$$

where $\otimes$ stand for the element-wise product. Similar solutions can be found for B and C using the other unfolded matrices. One can then alternately learn each of these matrices until convergence, see algorithm 5.

However this algorithm can not be directly applied to their framework. Indeed the number of items in each domain is inconsistent. To solve this issue they introduce a column-wise orthonormal matrix for each domain of size the number of items in the domains times the rank of the tensor. For example let consider domain k with sparse matrix $X_k \in \mathbb{R}^{n\times p_k}$, that can be factorize into $X_k \approx U\Sigma_k V_k^T$. Using $P_k \in \mathbb{R}^{p_k\times K}$ we can reformulate the factorization as $X_k \approx U\Sigma_k(P_k V)^T$, which also introduce the matrix $V \in \mathbb{R}^{K\times K}$ that does not vary by slice. Thus looking at the factorization of each $X_k, k \in [\kappa]$ with this formulation is equivalent to a PARAFAC decomposition:

$$\min_{U,V,\Sigma,P_k} \sum_{k=1}^{\kappa} ||X_k - U\Sigma_k(P_k V)^T||^2 \Leftrightarrow \min_{U,V,\Sigma,P_k} \sum_{k=1}^{\kappa} ||X_k P_k - U\Sigma_k V^T||^2 \quad (3.39)$$

Algorithm 5 CP-ALS-R($X, \lambda_A, \lambda_B, \lambda_C$).

Input: $\mathbf{X}$ the tensor to factorize, $\lambda_A, \lambda_B, \lambda_C$, the regularization parameters, the rank constraint K
Initial (sparse) A, B, C
while not converged **do**
  Fix B, C, update A according to equation 3.38
  Fix A, C, update B according to equation 3.38 replacing (1) by (2)
  Fix A, B, update C according to equation 3.38 replacing (1) by (3)
end while

Algorithm 6 HOSVD for tensor factorization.

Input: dataset $\mathcal{D}$, desired rank $\{k_d\}_{d=1}^{3}$
Construct the initial (sparse) 3-order tensor $\mathbf{X}$ from $\mathcal{D}$
for $d \in \{1, 2, 3\}$ *(modes)* **do**
  Compute the matrix unfolding $X_{(d)}$
  Apply SVD on $X_{(d)}$ and select the k_d largest singular values

$$\hat{X}_{(d)} = U^{(d)} \tilde{S}_d V_d^{\top}$$

end for
Compute $\mathbf{W} \doteq \mathbf{X} \times_1 U^{(1)\top} \times_2 U^{(2)\top} \times_3 U^{(3)\top}$
Return full tensor $\hat{\mathbf{X}} \doteq \mathbf{W} \times_1 U^{(1)} \times_2 U^{(2)} \times_3 U^{(3)}$

with $X_k P_k \in \mathbb{R}^{n \times K}$. Writing $\mathbf{Z}$ the corresponding $n \times K \times \kappa$ tensor where each slice corresponds to a $X_k P_k$ and define $C \in \mathbb{R}^{K \times K}$ such that $\Sigma_k \doteq \text{diag}(C_k)$, one can rewrite equation (3.39) as a PARAFAC decomposition following equation (3.36) (with $A = U$ and $B = V$) and apply CP-ALS-R. CP-ALS-R has the advantage to be simple and easily implemented. However, this approach suffers from major limitations. First, the definition of the least-squares problem does not deal with missing values and the tensor is assumed to be full. In recommender systems, this means that the initial sparse tensor is usually filled with zeros or the mean over each mode which can also cause memory issues. Furthermore, CP-ALS-R directly inherits one of the major drawbacks of the CP decomposition: each rank constraint must be the same for each mode. When the dimensions of the tensor are highly unbalanced, as in the case of e-commerce or restaurants, where we may have millions of users, thousand of products but only few actions, the rank constraint is either far too high for the action dimensions or far too low for the other modes preventing the model from learning appropriately.

Tucker decomposition without missing data.

One of the most popular approach to perform Tucker decomposition is the generalization of SVD to n-order tensors called HOSVD [Lathauwer et al., 2000], for which multiple versions have been proposed. We recall that the SVD of a matrix is the product of three matrices U, S and V, where U and V are orthonormal matrices of the left and right singular values and S is the diagonal matrix of (ordered) singular values of X such that $X = USV^\top$. In recommender systems, we preserve only the k largest singular values to get an approximation of X defined as $\hat{X} \doteq U\tilde{S}V^\top$, where $\tilde{S}$ is the truncated value of S where we replaced the $K+1, \cdots, \min(n,p)$ largest singular value by 0, thus $\hat{X}$ is the orthogonal projection of X on the space of rank-K matrices.

2- HOSVD In order to compute the Tucker decomposition, [Symeonidis et al., 2008] proposed to apply SVD on unfolded versions of the tensor as illustrated in algoritm 6. Once the SVD of each matrix is obtained, the core tensor is approximated by reversing the formula: $\mathbf{W} = \mathbf{X} \times_1 {U^{(1)}}^\top \times_2 {U^{(2)}}^\top \times_3 {U^{(3)}}^\top$. Their application focus on tag recommendation that is the first dimension encode users, the second encode the items and the third dimensions the tags. Each time a user give a tag to an item, the corresponding entry of the tensor is filled with a 1 whereas the missing values are implicitly translated into 0 entries. In many recommender systems, this may significantly bias the result and return poor prediction performance like SVD for matrix factorization.

Dealing with missing values.

Both previous tensor factorization algorithms can be adapted to take into consideration missing values. Let $\mathbf{P}$ a binary 3-order tensor defined as (see [Acar et al., 2010], [Tomasi and Bro, 2005])

$$\mathbf{P}_{ijk} = \begin{cases} 1, & \text{if } (i,j,k) \in \mathcal{D} \\ 0, & \text{else} \end{cases}$$

3- CP-WOPT Using $\mathbf{P}$, [Acar et al., 2010] adapts the objective function of the CP decomposition as

$$\begin{aligned} f_{\mathbf{P}}(A,B,C) &= \sum_{i,j,k=1}^{I,J,K} \left(\mathbf{P}_{ijk} \left(\mathbf{X}_{ijk} - \sum_{r=1}^{R} A_{ir} B_{jr} C_{kr} \right) \right)^2 \\ &= ||\mathbf{Y} - \mathbf{Z}||^2 \end{aligned}$$

with $\mathbf{Y} = \mathbf{P} * \mathbf{X}$ and $\mathbf{Z} = \mathbf{P} * [A,B,C]$, where $*$ represent the element by element multiplication. They propose to use nonlinear conjugate gradient method after having vectorized the two tensors (by concatenating all the modes into a long vector). The resulting algorithm has the advantage to use a simple first-order optimization method. Nevertheless it suffers from different drawbacks. First, this approach can be really memory greedy since for each step of the algorithm very long vectors with all the known values need to be stored, while for each update, HOALS store only the values related to a particular user (resp. item, or action), and thus fail in large recommender systems. Indeed, they show experiments on a $64 \times 4392 \times 28$ tensor that is very small compared to state-of-the-art recommender systems like Amazon or Netflix. Second, the algorithm proceeds sequentially and it can not be parallelized. This is also a major drawback in recommender systems that deal with millions of user and entries that cannot be managed by a unique non-parallel method. Having very time consuming learning may not be dramatic for some machine learning application like image recognition with neural network as long as the prediction is very effective, but for recommender systems users arrive everyday, items are added everyday and thus feedback are continuously provided. One may not have to update the model online after one feedback is acquired (however it could be very valuable for cold-start user/item) but should update it very frequently (in practice every night) in order to prevent user from being recommended already seen or purchased items and improve recommendation accuracy with those fresh information.

A similar approach can be followed for the Tucker decomposition by defining the objective

$$f_{\mathbf{P}}(\mathbf{W}, U, I, A) = \frac{1}{2}||\mathbf{P} * (\mathbf{X} - \mathbf{W} \times_1 U \times_2 I \times_3 A)||_F^2$$

4- Tucker decomposition with missing data In order to learn the decomposition, [Filipović and Jukić, 2013] used nonlinear conjugate gradient with HOSVD initialization. The gradients of the objective function are (similar for I and A)

$$\begin{aligned}
\nabla_U f_{\mathbf{P}} &= [\mathbf{P} * (\mathbf{W} \times_1 U \times_2 I \times_3 A - \mathbf{X})]_{(1)} \cdot \\
&\quad [(\mathbf{W} \times_2 I \times_3 A)_{(1)}]^\top \\
\nabla_{\mathbf{W}} f_{\mathbf{P}} &= \{\mathbf{P} * (\mathbf{W} \times_1 U \times_2 I \times_3 U - \mathbf{X})\} \times_1 \\
&\quad U^\top \times_2 I^\top \times_3 A^\top
\end{aligned}$$

Although interesting, this method has several limitations when applied to recommender systems. First, it does not include any regularization in the objective function and this may lead to overfitting data. Second, as for the CP models, the algorithm cannot be parallelized and this hinders its applicability to large scale recommender systems.

Finally, when dealing with multi-relational data, the problem can be seen as a particular tensor completion problem where each slice is a squared matrix.

5- RESCAL [Nickel et al., 2011] studied this specific problem by leveraging this particularity of the tensor in their RESCAL algorithm. Let consider the tensor $\mathbf{X} \in \mathbb{R}^{n \times n \times p}$, where for instance each slice indicate whether or not two users shared the same attribute (hair color, occupation, friendship, ..) and unknown values are set to 0. This particular problem lead the authors to consider a rank-K factorization of each slice $\mathbf{X}_k$ such that

$$\mathbf{X}_k \approx A R_k A^T, \forall k \in [p] \tag{3.40}$$

with $A \in \mathbb{R}^{n \times K}$ is common to each slice and contains the latent component of the users and $R_k \in \mathbb{R}^{K \times K}$ is an arbitrary squared matrix that weights the latent components for the k-th attribute. The matrices A and $(R_k)_{k \in [p]}$ are then learned in order to minimize the regularized squared loss:

$$\min_{A, R_k} \sum_k ||\mathbf{X}_k - A R_k A^T||_F^2 + \lambda(||A||_F^2 + \sum_k ||R_k||_F^2 \tag{3.41}$$

which is optimize following an alternating least squares approach. Although the authors filled the missing values by 0, one could derived a modified version that only minimize the error on the known entries. However interesting we could not compare with this approach because of the squared matrix constraint that make no sense in our case. We could use the same trick as [Hu et al., 2013] and detailed above in order to make the rectangular tensor square but it would make no sense to force users and items which have the same arbitrary index to have the same latent vectors.

3.6.3 Transfer Learning

Our work can also be related to transfer learning ([Pan and Yang, 2010]) which aims at transferring knowledge learned in some task in order to improve learning of a new task. This broad concept can served different purposes like meta-learning or multi-task learning and has been widely studied in supervised learning ([Mitov and Claassen, 2013],[Bakker and Heskes, 2003],[Evgeniou et al., 2005]) and reinforcement learning ([Wilson et al., 2007, Lazaric, 2008, Lazaric et al., 2008, Lazaric and Ghavamzadeh, 2010]). Transfer learning also receive attention in recommender systems. In order to fill the matrix $R_{(t)}$ of a target domain, [Pan et al., 2010] assumed the existence of two dense auxiliary matrices, $R_{(1)}$ and $R_{(2)}$ which are similar to the target domain, that is $R_{(t)}$ and $R_{(1)}$ share the same users whereas $R_{(t)}$ and $R_{(2)}$ share the same items. In order to approximate the factorisation of the target matrix $R_{(t)} \doteq U_{(t)} B_{(t)} V_{(t)}^T$, their algorithm called Coordinate System Transfer (CST) first learn the decomposition of $R_{(i)} \doteq U_{(i)} B_{(i)} V_{(i)}^T, i \in \{1, 2\}$ using classic matrix factorization techniques on sparse data in order to obtain good starting point to learn the factorization of $R_{(t)}$: $U_0 = U_{(1)}$ and $V_0 = V_{(2)}$, where U_0 and V_0 are the starting point in the learning of $U_{(t)}$ and $V_{(t)}$. U_0 and V_0 are also used to regularize the objective function - the square loss on the known entries - by adding a penalty term of the form $\lambda||U - U_0||_F^2 + \lambda||V - V_0||_F^2$ in order to force the algorithm to output values for $U_{(t)}$ (respectively $V_{(t)}$) that are in the ball of center U_0 (respectively V_0) which are learn alternatively together with $B_{(t)}$ with alternating gradient descent until convergence. If the domains are indeed correlated, the latent factors of $U_{(1)}$ and $V_{(2)}$ does contains good approximations of the one $U_{(t)}$ and $V_{(t)}$ that simplifies the learning of the target matrix. Similarly, the approach introduced in [Li et al., 2009a] assumes the existence of a related dense source but it is not assumed that source and target domains actually share the same user nor item which prevent from transfering user and item latent factors. They introduce the notion of *Codebook: Codebook is a $k \times l$ matrix ($k, l \leq n, p$) which compresses the cluster-level user-item rating patterns of k user clusters and l item clusters in the original rating matrix.* In other word the objective is to find a bi-clustering of user and item of the source domain into a small matrix, that contains user cluster-item cluster affinities in order to transfer it to the target domain that will leverage this new information to learn the decomposition. This codebook B is create from the orthogonal nonnegative matrix tri-factorization [Ding et al., 2006] of the source matrix $R_{(s)} \approx U_{(s)}S_{(s)}V_{(s)}^T$ with $U_{(s)} \in rr_+^{n\times k}, V_{(s)} \in rr_+^{p\times l}$, and $S_{(s)} \in rr_+^{k\times l}$ such that $U_{(s)}^T U_{(s)} = V_{(s)}^T V_{(s)} = I$, note that by definition each row of $U_{(s)}$ (resp. $V_{(s)}$) can only have one entry equal to 1 and the other equal to 0 which indicate the cluster the user (resp. item) belongs to. Thus,

$$B = [U_{(s)}^T R_{(s)} V_{(s)}] \oslash [U_{(s)}^T \mathbf{1}\mathbf{1}^T V_{(s)}] \tag{3.42}$$

where $\oslash$ means entry-wise division. The simplify objective of the target domain will then be to factorize the target matrix $R_{(t)}$ using the codebook B:

$$\min_{U,V} ||[R_{(t)} - UBV^T] \circ W||_F^2 \quad \text{such that } U\mathbf{1} = V\mathbf{1} = \mathbf{1} \tag{3.43}$$

where W is the binary matrix that worth 1 on the known entries and 0 elsewhere. In words, equation (3.43) look to which cluster each target user and item belong to in order to predict the missing values. When Q dense sources are available, [Moreno et al., 2012] extend codebook transfer method by computing a codebook $(B_q)_{q\in[Q]}$ for each source and look for a mix of them in the target domain:

$$\min_{U_q,V_q,\alpha_q} ||[R_{(t)} - \sum_{q=1}^{Q} \alpha_q U_q B_q V_q^T] \circ W||_F^2 \quad \text{such that } \forall q \in [Q],\ U_q\mathbf{1} = V_q\mathbf{1} = \mathbf{1} \tag{3.44}$$

Finally, [Li et al., 2009b] further developed a similar idea using projection of a hyperstructure that is transferred in order to capture non-linear correlation between domains.

3.6.4 Cold-start

Using multiple type of interaction has also the advantage to increase the number of feedback available per user and item, especially when among the different interaction only one is of interest for the selection choice. The cold-start problem have been studied along two directions. The first consist in relying on other source of information like using images or textual information and construct hybrid models ([Grbovic et al., 2016, Mikolov et al., 2013b, Le and Mikolov, 2014, Vasile et al., 2016]). Transfer learning introduced above can also be considered as tackling the cold-start issue by transfering users (respectively items) latent factors from an other application with the same users to the current task where users (respectively items) are in a cold-start situation. However in practice this situation almost never happens especially with the GDPR reinforcement law on data sharing between brands. Sharing personal data and personal feedback on items between brands is usually not possible. Thus such approaches are only possible for huge brand like Amazon and Wallmart who can "transfer" users information from different categories of their own brand. Other techniques to gather knowledge on new users consist in asking questions to users ([Elahi et al., 2013]) but asking effort to new user could increase the rebound rate, that is the number of users leaving the website after only seeing one page. Finally sometimes it is possible to have access to social graph in order to improve learning in cold-start situation. For instance [Sedhain et al., 2014] used social graph information to first compute similarity between cold-start and non cold-start users and then leverage non cold-start users prediction to output cold-start users prediction. Graph regularized matrix factorization can also be used when social information are available ([Cai et al., 2011]).

The second direction to solve cold-start problem is to apply exploration-exploitation methods that perform recommandation both to collect valuable feedback for future training and to satisfy users. Those question are related to bandit learning (see subsection 2.1.3) and best arm identification problems ([Soare et al., 2014, Jamieson and Talwalkar, 2016]). However application to collaborative filtering is not trivial. Indeed in linear bandit the output is a linear combination between a fix vector x that encode item information and a vector θ to learn. Even though in collaborative filtering the output is also a linear combination between two vectors here both vectors need to be learned and thus classic linear bandit proof can not be applied any more. A second difficulty is that in most recommender system application, one item can not be recommended multiple time to the same users, thus it is impossible to compute a confidence interval around this entry by pulling it multiple times. Aware of those limits, [Mary et al., 2014] propose a version where each entry can be sampled multiple times, for instance in music application or retargeting and focus on one cold-start: either user cold-start or item cold-start. For instance, in the case of user cold-start, they assume a sufficient number of feedback for each item thus the uncertainty on U is much more important than on V and the problem can be expressed using the linear bandit theory. For instance let consider user u and consider the $K \times K$ matrix A:

$$A = V_{I_u}^T V_{I_u} + \lambda n_u I$$

constructed with the latent factors of the items previously rated by user u. Then the UCB strategy select next item j following:

$$j \doteq \arg\max_{j \in I_u} U_u \cdot V_j^T + \alpha \sqrt{V_j A^{-1} V_j^T} \tag{3.45}$$

where $\alpha \in \mathbb{R}$ is an exploration parameter to be tuned.

The simple ε-greedy algorithm can also be applied to matrix factorization as done by [Guillou et al., 2016]. In this case, the matrix decomposition is periodically recomputed, and $\varepsilon/f(t)\%$ of the time, with $f(t)$ some fonction increasing with time like $f(t) = t$ or $f(t) = \log t$, an item is picked at random to be recommended to the active user. This modified version of the ε-greedy algorithm is called ε_n-greedy algorithm where ε is decreasing over time in order to allow lots of exploration at the beginning of the user journey and less and less over time when the system start to know the user very well. This algorithm called *SeALS* for Sequential ALS-WR experimentally proved to reduce the cumulative regret on several datasets like Movielens and Yahoo.

An other direction is proposed by [Li et al., 2016] that derived a sort of co-clustering algorithm with bandit techniques. In their framework, each item is defined by a known vector $x \in \mathbb{R}^d$, and the affinity between user u and item i of vector x is the scalar product between the user unknown vector $U_u \in \mathbb{R}^d$ and x plus sum noise: $r_{ui}(x) = U_u^T x + \varepsilon_u(x)$. In simple words, in order to estimate the affinity between the active user and every available items, the system will rely on a user clustering that depend on each item (all users that have close predicted affinity on this item will form a cluster) to compute an average estimate of U_u and of the V_u matrix (see Theorem 7) of past observation to compute a UCB reward. After choosing an item and observing the feedback, the system will update the clusters for the next step. This formula allows to leverage information from other users in order to alleviate from cold-start issues.

3.7 Industrial learnings

From an industrial point of view, we exploited HOALS in several applications at FIFTY-FIVE. Two applications were realised for a coupon/offer reduction recommendation problem for a food and a mall industry. In both cases the number of available coupons were not very large compared to classic e-commerce application, and users contextual information not reliable becauses users had to register online and often provided fake information or only an e-mail. Thus content-based models were not really an option. Users can interact with coupons in several ways. For the food industry the actions were the possibility to download on the website, download from an e-mail from a previous campaign a coupon, and finally actually purchase something using those coupons. For the mall industry there are clicks on the website page, clicks on the mobile application, Facebook likes and finally purchase. Classic matrix factorization was also not really an option because the real target action - the purchase - was a really rare event. HOALS was thus an interesting alternative and put into production for the campaign. Introduction of other dimensions like season of the year were considered for next steps of the activation. HOALS has also been used in order to create embedding of retailers for the mall industry to get insight on retailer correlations. Without HOALS it would have been difficult to get a global embedding of items since we would either have to (1) create an embedding for the mobile application based on mobile clicks and an other embedding based on the clicks on the website or (2) using only conversion. With the first solution, the two embeddings can be aggregated (sum or concatenated) but with HOALS all the feedback is used to compute one emdebding which is thus learned with more feedback than each separate embedding of this first solution and thus is more reliable. The second solution was not possible because conversions were very rare in the dataset, since most online users are not linked to offline conversions. Finally, using bandit methods to solve the cold-start problem although interesting in theory suffer from several industrial constraints. The first is that bandit learning are useful to achieve good long term performance but most of the time one cannot wait for long term and a recommender system must leverage very few user sessions. Thus using all information provided by each user during a session like click, add to cart, conversion, zoom, reading of comments or time on the page is often a better approach than applying linear bandit using only the conversion as a target to be maximized. Today, in order to work around the cold start issue, the trend is to collect feedback using new technologies to interact and engage users like chatbot or virtual/augmented reality.

3.8 Conclusion

In this chapter we introduced a parallel tensor factorization algorithm for recommender systems built on an extension of alternating least squares. For e-commerce, the purchase action is the most preferable action to predict but also the most difficult for matrix factorization given its high sparsity. However, our HOALS model proved to be effective to predict any action by leveraging knowledge from the other actions and thus make it an efficient transfer learning algorithm for RSs. An important advantage of HOALS over state-of-the-art methods such as CP-ALS-R and HOSVD is its high ability to be parallelized, and thus its feasible for large RSs dealing with millions of users and items. For instance, for a n-order tensor, since CP-ALS-R computes matrices one after the other, HOALS is n times more parallelizable. We also extended the implicit feedback strategy introduced in [Hu et al., 2008] to tensor factorization, making our algorithm a very effective approach to solve efficiently RSs with implicit and multiple feedback. Finally, HOALS can also be used to create embedding of users or items using all available feedback at the same time in order to be integrated into an hybrid recommender system or provide clustering information.

Basket completion with multi-task Determinantal Point Process

Determinantal point processes (DPPs) have received significant attention in the recent years as an elegant model for a variety of machine learning tasks, due to their ability to model set diversity and item quality or popularity. Recent works have shown that DPPs can be effective models for product recommendation and basket completion tasks. We present an enhanced DPP model that is specialized for the task of basket completion, the Multi-Task DPP. We view the basket completion problem as a multi-task classification problem, and leverage ideas ideas from tensor factorization and multi-class classification to design the Multi-Task DPP model. We evaluate our model on several real-world datasets, and find that the Multi-Task DPP provides significantly better predictive quality than a number of state-of-the-art models.

4.1 Introduction

Increasing the number of items in the average shopping basket is a major concern for online retailers. While there are a wide range of possible strategies, this work focuses on the algorithm responsible for proposing a set of items that is best suited to complete the current shopping basket of the user.

Basket analysis and completion is a very old task for machine learning. For many years association rule mining [Agrawal et al., 1993, Hipp et al., 2000, Liu et al., 1998] has been the state-of-the-art. Even though there are different variants of this algorithm, the main principle involves computing the conditional probability of buying an additional product by counting co-occurrences in past observations. Because of the computational cost and robustness, modern approaches favor item-to-item collaborative filtering [Linden et al., 2003], or using logistic regression to predict if a user will purchase an item based on binary purchase scores obtained from market baskets [Lee et al., 2005].

As reported in related work, vanilla collaborative filtering needs to be extended to correctly capture diversity among products. Pracitioners often mitigate this problem by adding constraints to the recommended set of items. As an example, when using categorical information it is possible to always recommend one pair of matching shoes when trousers are added to the basket, even if natural co-sales could lead to the recommendation of other trousers. In this situation the presence of diversity in the recommendations is not directly driven by the learning algorithm, but by side information and expert knowledge. Ref. [Teo et al., 2016] proposes an effective Bayesian method for learning the weights of the categories in the case of visual search when categories are known.

Sometimes we need to learn diversity without relying on extra information. Naive learning of diversity directly from the data without using side information comes at a high computational cost because the number of possible sets grows exponentially with the number of items. The issue is not trivial, even when we want to be able to add only one item to an existing set, and becomes even harder when we want to add more than one item with the idea of maximizing the diversity of the final recommended set.

Refs. [Gartrell et al., 2017, Gartrell et al., 2016] address this combinatorial problem using a model based on Determinantal Point Processes (DPPs) for basket completion. DPPs are elegant probabilistic models of repulsion from quantum physics, which are used for a variety of tasks in machine learning [Gautier et al., 2017, Kulesza and Taskar, 2012, Foulds and Görür, 2013]. They allow sampling a diverse set of points, with similarity being encoded using a positive semi-definite matrix called the kernel. Efficient algorithms for marginalization and conditioning DPPs are available. From a practical perspective, learning the DPP kernel is a challenge because the associated likelihood is not convex, and learning it from observed sets of items is conjectured to be NP-hard [Kulesza and Taskar, 2012].

For basket completion it is natural to consider that sets are the baskets which converted to sales. In this setting, the DPP is parametrized by a kernel matrix of size $p \times p$, where p is the size of the catalog. Thus the number of parameters to fit grows quadratically with the number of items and the computational complexity cublicly. As learning a full-rank DPP is hard, [Gartrell et al., 2017] proposes regularizing the DPP by constraining the kernel to be low rank. This regularization also improves generalization and offers more diversity in recommendations without hurting predictive performance. In fact in many settings the predictive quality is also improved, making the DPP a very desirable tool for modeling baskets. Moreover, the low rank asumption also enables substantially better runtime performance compared to a full-rank DPP.

Nevertheless, because of the definition of DPP, as described in Section 4.3, this low-rank assumption for the kernel means that any possible baskets with more items than the chosen rank will receive a probability estimation of 0. This technique is thus impossible to use for large baskets, and some other regularizations of the DPP kernel can be more intersting. The contribution of this chapter is threefold:

- We modify the constraints over the kernel to support large baskets.
- We model the probability over all baskets by adding a logistic function on the determinant computed from the DPP kernel. We adapt the training procedure to handle this nonlinearity, and evaluate our model on four popular basket datasets.
- By leveraging tensor factorization we propose a new way to regularize the kernel among a set of tasks. This approach also leads to enhanced predictive quality.

We also show that these ideas can be combined for further improvements to predictive quality, allowing our Multi-Task DPP model to outperform state-of-the-art models by a large margin.

In the next section we introduce Determinantal Point Processes in order to give the appropriate background before presenting our proposed algorithm in the following section which effectiveness is validatating on various real world dataset in the next section. We then discuss related work before concluding and introducing possible future works.

4.2 Background

Generally a Point Process $\mathbb{P}$ is a probability measure over a set of points, and Determinantal Point Processes (DPPs) compute these probabilities using determinants. DPPs were originally used to model a distribution over particles that exhibit a repulsive effect [Vershik and Yakubovichn, 2001]. Recently, interest in leveraging this repulsive behavior led to DPPs receiving increasing attention within the machine learning community. Mathematically, discrete DPPs are distributions over discrete sets of points (note that DPPs can be defined for the continuous case), or in our case items, where the model assigns a probability to observe a given set of items $\mathcal{I} \subset [p] \doteq [1, \cdot, p]$, with p the number of items. A process $\mathbb{P}$ is called a determinantal point process if it follows definition 4.

Definition 4 (Determinantal Point Process). *If for all Y random subsets drawn according to $\mathbb{P}$, we have*

$$\forall \mathcal{I} \subset [p], \ \mathbb{P}(\mathcal{I} \subset Y) = \det(K_{\mathcal{I}}) \tag{4.1}$$

with $K \in \mathbb{R}^{p \times p}$ a real symmetric positive semidefinite matrix, then $\mathbb{P}$ is a Determinantal Point Process. $K_{\mathcal{I}}$ denotes the principal submatrix of K indexed *by the elements in $\mathcal{I}$.*

A DPP is then a probability measure on 2^p (the power set, or set of all subsets of p). However in practice it is more convenient to define a DPP through L-ensembles, that defines a DPP according to a real, symmetric matrix L, that directly output the probability to observe a given set $\mathcal{I}$:

$$\mathbb{P}(\mathcal{I}) \propto \det L_{\mathcal{I}} \tag{4.2}$$

The equality is obtained by dividing the right part of equation 4.2 by $\det(L+I)$, because it can be shown that $\sum_{\mathcal{I}} \det L_{\mathcal{I}} = \det(L+I)$ (see Theorem 2.1 of [Kulesza and Taskar, 2012]) such that $\mathbb{P}(\mathcal{I}) = \det L_{\mathcal{I}} / \det(L+I)$. Those two definitions are equivalent and one can show that $K = L(L+I)^{-1}$ (see Theorem 2.2 of [Kulesza and Taskar, 2012]). The repulsive behaviour of DPPs can be easily seen in the particular case of selecting two items i and j together, since this probability is

$$\mathbb{P}[\{i,j\}] \propto \begin{vmatrix} L_{ii} & L_{ji} \\ L_{ji} & L_{jj} \end{vmatrix} = \mathbb{P}[\{i\}]\mathbb{P}[\{j\}] - L_{ij}^2 \tag{4.3}$$

In equation 4.3 we can see that the more similar i and j are - that is the higher L_{ij} is - the less likely they are to be sampled together. The definition of the entries L_{ij} will therefore determine the replusive behavior of the kernel for the task. For instance, if similarity is defined using image descriptors such that $L_{ij} = V_i V_j^T$ with $V_i \in \mathbb{R}^K$ the embedding of item i, then images of differing appearance will be more likely to be sampled by a DPP. On the other hand, if the entries L_{ij} are learned using previously observed sets, such as e-commerce baskets [Gartrell et al., 2017], then co-purchased items i and j will are likely to be sampled by the DPP, and thus the "similarity" L_{ij} will be low. In an application such as a search engine or in document summarization, the kernel may be defined using feature descriptors $V_i \in \mathbb{R}^D$ (i.e tf-idf of the text with D the size of the dictionary), and a relevance score $q_i \in \mathbb{R}^+$ of each item i, such that $L_{ij} = q_i V_i^T V_j q_j$, which favors relevant items (large q_i) and discourages lists composed of similar items.

We see that kernel of the DPP can either be explicitly defined in order to induce diversity in future samples, or be learned in order to measure set correlation in some data. In the following we will focus on the later for the basket completion application.

4.3 Model

4.3.1 Logistic DPP

Our objective is to find a set of items that are most likely to be purchased together. We formulate this as a classification problem, where the goal is to predict if a specific set of items will generate a conversion from the user, which we denote as $Y \in \{0, 1\}$.

We model the class label Y as a Bernoulli random variable of parameter $\phi(\mathcal{I})$, where $\mathcal{I}$ is the set of items and ϕ is a function that we will define later:

$$p(y|\mathcal{I}) = \phi(\mathcal{I})^y (1 - \phi(\mathcal{I}))^{1-y} \tag{4.4}$$

We model the function ϕ using a DPP.

We assume that there exists a latent space such that diverse items in this space are likely to be purchased together. Similarly to [Gartrell et al., 2017], we assume a low-rank factorization of the kernel matrix $L \in \mathbb{R}^{p \times p}$ according to the following definition:

Definition 5 (L kernel matrix of Logistic DPP).

$$L = VV^T + D^2 \tag{4.5}$$

where $V \in \mathbb{R}^{p \times r}$ is a latent matrix where each row vector i encodes the r latent factors of item i. D is a diagonal matrix that, and together with $||V_i||$, represents the intrinsic quality or popularity of each item.

The squared exponent on D in definition 5 insures that we always have a valid positive semi-definite kernel. We then define $\phi(\mathcal{I}) \propto \det(V_{\mathcal{I},:} V_{\mathcal{I},:}^T + D^2) \geq 0$. Note that without the diagonal term, the choice of r would restrict the cardinality of the observable set, because $|\mathcal{I}| > r$ would imply $\phi(\mathcal{I}) = 0$ when $D \equiv 0$. Using this term will ensure that the success probability of any set will be positive, but the cross-effects will be lower for sets of cardinality higher than r. We also see that items with similar latent vectors are less likely to be sampled than items with different latent vectors, since similar vectors will produce a parallelotope with a smaller volume. To normalize the probability and encourage separation between vectors we define Φ according to a logistic function as follow:

Definition 6 (Success probability of Logistic DPP). *Let L be defined according to definition 5*

$$\phi(I) = \mathbb{P}(y = 1|\mathcal{I}) \doteq 1 - \exp(-w \det L_{\mathcal{I}}) \tag{4.6}$$
$$\doteq \sigma(w \det L_{\mathcal{I}}) \tag{4.7}$$

Usually the logistic function is of the form $\sigma(x) = 1/(1+\exp(-x))$. However, in our case the determinant is always positive since L is positive semi-definite, which would result in $\mathbb{P}(y = 1|\mathcal{I})$ always greater than 0.5 with such a function. By construction, our formulation allows us to obtain a probability between 0 and 1. Finally, $w \in \mathbb{R}$ is a scaling parameter to be learned by cross-validation that insures that the exponential does not explode. Indeed, popular items will have a value $L_{ii} > 1$ in order to increase the determinant when in a set. Consequently the determinant of a set containing multiple independant popular items would be of the order of α^q with $\alpha > 1$ and q the number of items and thus could explode.

Learning. In order to learn the matrix V we assume the existence of historical data $\{\mathcal{I}_m, y_m\}_{1\leq m\leq M}$, where $\mathcal{I}_m$ is a set of items, and y_m is a label set to 1 if the set has been purchased and 0 otherwise. This training data allows us to learn the matrices V and D by maximizing the log-likelihood of the data. To do so, we first write the click probability for all y as

$$\mathbb{P}(y|\mathcal{I}) = \sigma(w \det L_{\mathcal{I}})^y (1 - \sigma(w \det L_{\mathcal{I}}))^{1-y} \tag{4.8}$$

The penalized log-likelihood $f(V, D)$ can then be written as

$$\begin{aligned}
f(V,D) &= \log \prod_{m=1}^{M} \mathbb{P}(y_m|\mathcal{I}_m) - \frac{\alpha_0}{2}\sum_{i=1}^{p} \alpha_i(||V_i||^2 + ||D_i||^2) && (4.9)\\
&= \sum_{m=1}^{M} \log \mathbb{P}(y_m|\mathcal{I}_m) - \frac{\alpha_0}{2}\sum_{i=1}^{p} \alpha_i(||V_i||^2 + ||D_i||^2) && (4.10)\\
&= \sum_{m=1}^{M} y_m \log \sigma(\det L_{\mathcal{I}_m}) + (1-y_m)\log(1-\sigma(\det L_{\mathcal{I}_m})) && (4.11)\\
&\quad -\frac{\alpha_0}{2}\sum_{i=1}^{p} \alpha_i(||V_i||^2 + ||D_i||^2)
\end{aligned}$$

Following [Gartrell et al., 2017], α_i is an item regularization weight that is inversely proportional to item popularity, that prevent from overfitting.
Optimization. We maximize the log-likelihood using stochastic gradient ascent with Nesterov's Accelerated Gradient, which is a form of momemtum. To simplify notation, we define $[m] \doteq \mathcal{I}_m$ and $\sigma_m = \sigma(w \det L_{[m]})$. Let $i \in \{1, \cdots, p\}, k \in \{1, \cdots, r\}$,

Lemma 2. *When D is fixed, the gradient of (4.9) with respect to V_{ik} is*

$$\frac{\partial f}{\partial V_{ik}} = 2w \sum_{m, i\in[m]} ([L_{[m]}^{-1}]_{:,i} V_{:,k}) \frac{y_m - \sigma_m}{\sigma_m} \det L_{\mathcal{I}_m} - \alpha_0 \alpha_i V_{ik} \tag{4.12}$$

Proof Without the regularization term we have

$$\begin{aligned}
\frac{\partial f}{\partial V_{ik}} &= \sum_{m,i\in[m]} \frac{y_m}{\sigma_m}\frac{\partial \sigma_m}{\partial V_{ik}} + \frac{1-y_m}{1-\sigma_m}\left(-\frac{\partial \sigma_m}{\partial V_{ik}}\right) && (4.13)\\
&= w \sum_{m,i\in[m]} \frac{y_m - \sigma_m}{\sigma_m} \frac{\partial \det L_{[m]}}{\partial V_{ik}} && (4.14)\\
&= w \sum_{m,i\in[m]} \frac{y_m - \sigma_m}{\sigma_m} \operatorname{tr}\left(L_{[m]}^{-1} \frac{\partial L_{[m]}}{\partial V_{ik}}\right) \det L_{\mathcal{I}_m} && (4.15)\\
&= 2w \sum_{m,i\in[m]} ([L_{[m]}^{-1}]_{:,i} V_{:,k}) \frac{y_m - \sigma_m}{\sigma_m} \det L_{\mathcal{I}_m} && (4.16)
\end{aligned}$$

where (4.16) follows from

$$\left[\frac{\partial L_{[m]}}{\partial V_{ik}}\right]_{s,t} = V_{sk}\delta_{i,t} + V_{tk}\delta_{i,s} \tag{4.17}$$

Therefore,

$$\begin{aligned}
\operatorname{tr}\left(L_{[m]}^{-1}\frac{\partial L_{[m]}}{\partial V_{ik}}\right) &= \sum_{s,t}(V_{sk}\delta_{i,t} + V_{tk}\delta_{i,s})[L_{[m]}^{-1}]_{s,t} && (4.18)\\
&= \sum_{s}[L_{[m]}^{-1}]_{s,i}V_{sk} + \sum_{t}[L_{[m]}^{-1}]_{i,t}V_{tk}\\
&= 2\sum_{s}[L_{[m]}^{-1}]_{s,i}V_{sk} && (4.19)
\end{aligned}$$

adding the derivative of the regularization concludes the proof. □

Lemma 3. *When V is fixed, the gradient of (4.9) with respect to D_i is*

$$\frac{\partial f}{\partial D_{ii}} = 2w\sum_{m,i\in[m]}([L_{[m]}^{-1}]_{i,i}D_{i,i})\frac{y_m - \sigma_m}{\sigma_m}\det L_{[m]} - \alpha_0\alpha_i D_{ii} \tag{4.20}$$

Proof As shown previously and without the regularization term we have

$$\frac{\partial f}{\partial V_{ik}} = w\sum_{m,i\in[m]}\frac{y_m - \sigma_m}{\sigma_m}\operatorname{tr}\left(L_{[m]}^{-1}\frac{\partial L_{[m]}}{\partial D_{i,i}}\right)\det L_{\mathcal{I}_m} \tag{4.21}$$

Since,

$$\left[\frac{\partial L_{[m]}}{\partial D_{i,i}}\right]_{s,t} = 2D_{i,i}\delta_{s,i}\delta_{t,i} \tag{4.22}$$

we see that

$$\operatorname{tr}\left(L_{[m]}^{-1}\frac{\partial L_{[m]}}{\partial D_{i,i}}\right) = 2[L_{[m]}^{-1}]_{i,i}D_{i,i} \tag{4.23}$$

adding the derivative of the regularization concludes the proof. □

4.3.2 Multi-Task DPP

We now propose a modification to the previously introduced model that is better suited for the basket completion task. To do so we enhance the Logistic DPP for the basket completion scenario, where we model the probability that the user will purchase a given set of items. Here we formulate this with a Multi-Task DPP, where the goal is to predict whether the user will purchase a given target item

Algorithm 7 Optimization algorithm for Logistic DPP model.

Input: $w \in \mathbb{R}$ the scaling weight, $\alpha_0 \in \mathbb{R}$, $\beta \in \mathbb{R}$ the momemtum coefficient, $b \in \mathbb{N}$ the minibatch size, $\varepsilon \in \mathbb{R}$ the gradient step, $t = 0$ the iteration counter, $T = 0$, past data $\mathcal{D} = \{\mathcal{I}_m, y_m\}_{1 \leq m \leq M}$.
Initialization: Compute item popularity and output regularization weights α_i.
Set $D_0 \sim \mathcal{N}(1, 0.01)$ on the diagonal and $\tilde{D}_0 \equiv 0$ the gradient accumulation on D.
Set $V_0 \sim \mathcal{N}(0, 0.01)$ everywhere and $\tilde{V}_0 \equiv 0$ the gradient accumulation on V.
while not converged **do**
  if $b(t+1) > M(T+1)$ **then**
    Shuffle $\mathcal{D}$ and set $T = T + 1$
  end if
  Update $(\tilde{V}_{t+1}, \tilde{D}_{t+1}) = \beta(\tilde{V}_t, \tilde{D}_t) + (1-\beta)\varepsilon \bigtriangledown f(V_t + \beta\tilde{V}_t, D_t + \beta\tilde{D}_t)$ according to formulas (4.12) and (4.20)
  Update $V_{t+1} = V_t + \tilde{V}_{t+1}$
  Update $D_{t+1} = D_t + \tilde{D}_{t+1}$
end while

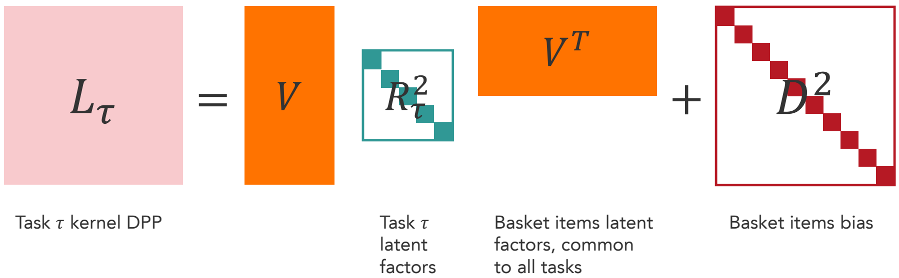


Figure 4.1: Multi-Task DPP factorization.

based on the user's basket. We think that such a model is more suitable for basket completion because it will capture *directed completion rules*. Indeed, in some cases it may be a good idea to recommend item B to basket containing item A, but not the other way around. See the *pillow-couch* example in the related work. In this setting, there are as many slices in the tensor as there are items in the catalog that is p. Learning one kernel per item would be impossible in practice and suffer from sparsity issues. Indeed, with one kernel per item, each target item would be present in only a fraction of the baskets, and thus dramatically reduce the size of the training set per kernel. To solve this issue we use a low-rank tensor point of view. We define a cubic tensor $L \in \mathbb{R}^{p \times p \times p}$, where each slice τ (noted L_τ) of L is the target item (low-rank) kernel. By assuming that the tensor L is low-rank, we are able to implement sharing of learned parameters between target items, as defined by the following equation:

Definition 7 (**L kernel matrix of Multi-Task DPP, figure 4.1**).

$$L_\tau = V R_\tau^2 V^T + D^2 \tag{4.24}$$

where $V \in \mathbb{R}^{p \times r}$ are the item latent factors that are common to all items, and $R_\tau \in \mathbb{R}^{r \times r}$ is a target specific matrix that models the interactions between the latent components of each target item. In order to balance the degrees of freedom between targets and items, we further assume that R_τ is a diagonal matrix. Therefore, the diagonal vector of R_τ models the latent factors of each target, and the latent factors of the item can be seen as the relevance of the product for each latent factor.

As in the case for the matrix D, the squared exponent on R_τ in definition 4.24 ensures that we always have a valid kernel. This decomposition is similar to the RESCAL decomposition ([Nickel et al., 2011], introduced in section 3.6.2) without the additional bias term and a diagonal constraint on the slice specific matrix. Besides we also have a different learning procedure due to the use of the logistic function. Finally, the probability that a set of items $\mathcal{I}$ will be successful for target τ is

$$\mathbb{P}(y_\tau = 1|\mathcal{I}) = \sigma(w \det L_{\tau,\mathcal{I}}) = 1 - \exp(-w \det L_{\tau,\mathcal{I}}) \tag{4.25}$$

Therefore, the log-likelihood is

$$\begin{aligned} g(V, D, R) &= \log \prod_m^M \mathbb{P}(y_\tau|[m]) - \frac{\alpha_0}{2} \sum_{i=1}^p \alpha_i (||V_i||^2 + ||D_i||^2 + ||R^i||^2) && (4.26) \\ &= \sum_m^M \log \mathbb{P}(y_\tau|[m]) - \frac{\alpha_0}{2} \sum_{i=1}^p \alpha_i (||V_i||^2 + ||D_i||^2 + ||R^i||^2) && (4.27) \end{aligned}$$

Since each observation m is attached to a task, we denote τ_m the task that corresponds to observation m. Thus we have $\sigma_m = \sigma(\det L_{\tau_m,[m]})$. When there is no ambiguity, we also denote $L_{[m]} \doteq L_{\tau_m,[m]}$. Let $i \in \{1, \cdots, p\}, k \in \{1, \cdots, r\}$,

Lemma 4. *When D and R are fixed, the gradient of (4.26) with respect to V_{ik} is*

$$\begin{aligned} \frac{\partial g}{\partial V_{ik}} &= 2w \sum_{m, i \in [m]} \frac{y_{\tau_m} - \sigma_m}{\sigma_m} R^2_{\tau_m,k,k} [L^{-1}_{\tau_m,[m]}]_{:,i} \cdot V_{:,k} \det L_{\tau_m,[m]} && (4.28) \\ & \quad - \alpha_0 \alpha_i V_{ik} \end{aligned}$$

Proof Without the regularization term we have

$$\frac{\partial g}{\partial V_{ik}} = \sum_{m,i\in[m]} \frac{y_{\tau_m}}{\sigma_m}\frac{\partial \sigma_m}{\partial V_{ik}} + \frac{1-y_{\tau_m}}{1-\sigma_m}\left(-\frac{\partial \sigma_m}{\partial V_{ik}}\right) \tag{4.29}$$

$$= w \sum_{m,i\in[m]} \frac{y_{\tau_m}-\sigma_m}{\sigma_m}\frac{\partial \det L_{[m]}}{\partial V_{ik}} \tag{4.30}$$

$$= w \sum_{m,i\in[m]} \frac{y_m-\sigma_m}{\sigma_m} \operatorname{tr}\left(L_{[m]}^{-1}\frac{\partial L_{[m]}}{\partial V_{ik}}\right) \det L_{[m]} \tag{4.31}$$

$$\tag{4.32}$$

Since,

$$[L_\tau]_{s,t} - D_{s,t}^2 = \sum_{j=1}^{r}[VR_\tau^2]_{s,j}V_{t,j} = \sum_{j=1}^{r} V_{s,j}R_{\tau,j,j}^2 V_{t,j} \tag{4.33}$$

we have

$$\left[\frac{\partial L_{[m]}}{\partial V_{ik}}\right]_{s,t} = R_{\tau,k,k}^2(V_{t,k}\delta_{s,i} + V_{s,k}\delta_{t,i}) \tag{4.34}$$

Thus,

$$\operatorname{tr}\left(L_{[m]}^{-1}\frac{\partial L_{[m]}}{\partial V_{ik}}\right) = 2R_{\tau,k,k}^2 \sum_{s\in[m]} [L_{\tau_m,[m]}^{-1}]_{s,i}V_{s,k} \tag{4.35}$$

adding the regularization concludes the proof. □

Lemma 5. *When V and R are fixed, the gradient of (4.26) with respect to $D_{i,i}$ is*

$$\frac{\partial g}{\partial D_{i,i}} = 2w \sum_{m,i\in[m]} \frac{y_{\tau_m}-\sigma_m}{\sigma_m}[L_{t_m,[m]}^{-1}]_{i,i}D_{i,i}\det L_{\mathcal{I}_m} - \alpha_0\alpha_i D_{ii} \tag{4.36}$$

Proof Similarly without the regularization term we have

$$\frac{\partial g}{\partial D_{i,i}} = w \sum_{m,i\in[m]} \frac{y_m-\sigma_m}{\sigma_m} \operatorname{tr}\left(L_{[m]}^{-1}\frac{\partial L_{[m]}}{\partial D_{i,i}}\right) \det L_{[m]} \tag{4.37}$$

Using (4.33)

$$\left[\frac{\partial L_{[m]}}{\partial D_{i,i}}\right]_{s,t} = 2D_{i,i}\delta_{s,i}\delta_{t,i} \tag{4.38}$$

thus

$$\operatorname{tr}\left(L_{[m]}^{-1}\frac{\partial L_{[m]}}{\partial D_{i,i}}\right) = 2[L_{t_m,[m]}^{-1}]_{i,i}D_{i,i} \tag{4.39}$$

adding the regularization concludes the proof. □

Lemma 6. *When V and D are fixed, the gradient of (4.26) with respect to $R_{k,k}$ is*

$$\frac{\partial g}{\partial R_{\tau,k,k}} = 2w \sum_{m,\tau\in[m]} \frac{y_\tau - \sigma_m}{\sigma_m} R_{\tau,k,k} L^{-1}{}_{[m]} \cdot V_{:,k} \cdot V_{:,k}^T \det L_{[m]} \quad (4.40)$$
$$-\alpha_0\alpha_\tau R_{\tau,k,k},$$

Proof Similarly without the regularization we have

$$\frac{\partial g}{\partial R_{\tau,k,k}} = w \sum_{m,i\in[m]} \frac{y_\tau - \sigma_m}{\sigma_m} \operatorname{tr}\left(L_{[m]}^{-1} \frac{\partial L_{[m]}}{\partial R_{\tau,k,k}}\right) \det L_{[m]} \quad (4.41)$$

$$(4.42)$$

Using (4.33)

$$\left[\frac{\partial L_{[m]}}{\partial R_{\tau,k,k}}\right]_{s,t} = 2R_{\tau,k,k} V_{s,k} V_{t,k} \quad (4.43)$$

we obtain

$$\operatorname{tr}\left(L_{[m]}^{-1} \frac{\partial L_{[m]}}{\partial R_{\tau,k,k}}\right) = 2R_{\tau,k,k} \sum_{s,t\in[m]} [L_{[m]}^{-1}]_{s,t} V_{s,k} V_{t,k} \quad (4.44)$$

adding the regularization concludes the proof. □

4.3.3 Prediction

As previsouly discussed, sampling from a DPP can be a difficult problem, and various solutions have been proposed [Han et al., 2017, Gillenwater et al., 2014, Gautier et al., 2017, Chen et al., 2017]. Although sampling the best set among all possible set has been conjectured to be NP-hard, our goal is to find only the best item to complete the basket. In such applications a greedy approach can be applied effectively, especialy with the low-rank structure of our model. In addition, [Gartrell et al., 2017] proposed an effective method for the basket completion scenario that involves conditioning the DPP, which can be applied to our Logistic DPP model.

4.4 Experiments

We evaluated the performance of our models on the basket completion problem on several real-world datasets, and compared it to several state-of-the-art baselines.

MODELS

- **Our models**. To understand the impact of the different components of our model compared to the low-rank DPP model, we evaluated the following versions of our model:

Algorithm 8 Optimization algorithm for Multi-Task DPP model.

Input: $w \in \mathbb{R}$ the scaling weight, $\alpha_0 \in \mathbb{R}$, $\beta \in \mathbb{R}$ the momemtum coefficient, $b \in \mathbb{N}$ the minibatch size, $\varepsilon \in \mathbb{R}$ the gradient step, $t = 0$ the iteration counter, $T = 0$, past data $\mathcal{D} = \{\mathcal{I}_m, y_m\}_{1 \le m \le M}$.
Initialization: Compute item popularity and output regularization weights α_i.
Set $D_0 \sim \mathcal{N}(1, 0.01)$ on the diagonal and $\tilde{D}_0 \equiv 0$ the gradient accumulation on D.
Set $V_0 \sim \mathcal{N}(0, 0.01)$ everywhere and $\tilde{V}_0 \equiv 0$ the gradient accumulation on V.
Set $R_{\tau,0} \sim \mathcal{N}(1, 0.01)$ on the diagonal for each task and $\tilde{R}_{\tau,0} \equiv 0$ the gradient accumulation on R_τ.
while not converged **do**
  if $b(t+1) > M(T+1)$ **then**
    Shuffle $\mathcal{D}$ and set $T = T + 1$
  end if
  Update $\left(\tilde{V}_{t+1}, \tilde{D}_{t+1}, (\tilde{R}_{\tau,t+1})_\tau\right) = \beta\left(\tilde{V}_t, \tilde{D}_t, (\tilde{R}_{\tau,t})_\tau\right) + (1-\beta)\varepsilon \nabla g(V_t + \beta\tilde{V}_t, D_t + \beta\tilde{D}_t, (R_{\tau,t} + \beta\tilde{R}_{\tau,t})_\tau)$ according to formulas (4.28), (4.36) and (4.40)
  Update $V_{t+1} = V_t + \tilde{V}_{t+1}$
  Update $D_{t+1} = D_t + \tilde{D}_{t+1}$
  Update $R_{\tau,t+1} = R_{\tau,t} + \tilde{R}_{\tau,t+1}$ for all τ
end while

- **Logistic DPP**: This version is similar to the low-rank DPP model, with the addition of the logistic function. To determine what item to recommend we use a greedy approach, where we select the next item such that the probability of the basket completed with this item is the largest.
- **Multi-Task log-DPP without bias**: In this version we set $D \equiv 0$, which allows us to measure the impact of capturing the item bias in a separate matrix. The matrix V encodes the latent factors of items present in the basket, while each matrix R_τ encodes the latent factors of each target item τ to add to a basket.
- **Multi-Task log-DPP**: This is the full version of our model, with bias enabled.

Our datasets do not provide explicit negative information. To generate negative feedback for our models we created negatives targets from observed baskets by sampling a random item among those items not in the basket. This approach could be improved through better negative sampling strategies, but since this is not part of our primary contributions we leave this investigation for future work.

- **Baselines**. The primary goal of our work is to improve state-of-the-art results provided by DPPs and introduce new modeling enhancements to DPPs. However, for the sake of completness we also compare with other strong baseline models provided by state-of-the-art collaborative filtering approaches.

- **Poisson Factorization** (PF) [Gopalan et al., 2013] is a probabilistic matrix factorization model generally used for recommendation applications with implicit feedback. Since our datasets contain no user id information, we consider each basket to be a different user, and thus there are as many users as baskets in the training set. In practice this can cause memory issues, since the number of baskets can be very large.
- **Factorization Machines** (FMs) [Rendle, 2010] is a general approach that models dth-order interactions using low-rank assumptions. FMs are usually used with $d = 2$, since this corresponds to classic matrix factorization, which is the value we use here. As with PF, to learn the FM we consider each basket as a unique user. For fairness in comparison with our models, we also tried a FM with negative sampling based on item popularity. However, we did not see any substantial improvement in model performance using this negative sampling technique.
- **Low-Rank DPP** [Gartrell et al., 2017] is a low-rank DPP model, suitable for basket completion and other tasks, where the determinant of the sub-kernel corresponds to the probability that all the items will be bought.
- **Bayesian Low-RankDPP** [Gartrell et al., 2016] is the Bayesian version of the low-rank DPP model.
- **Associative Classifier** (AC), detailed in section 4.5.1, is a algorithm that computes support of bought set items in order to obtain completion rules. As [Gartrell et al., 2017], we used the Classification Based on Associations (CBA) algorithm [Liu et al., 1998], available at [Coenen, 2005], with minimum support of 1.0% and maximum confidence thresholds of 20.0%. Unlike other models, AC do not provide estimates for all possibles sets. Thus some metrics can not be computed like the MPR, detailed below, that we will use.
- **Recurrent Neural Network** (RNN) [Hidasi et al., 2015] adapted for session-based recommender system. RNN requires ordered sequences, thus we will only test this model performance on our dataset where the ordering of the purchases within each basket is available. We use the code of [Songweiping, 2017].

In the interest of reproducibility, all model implementations codes are available at [Warlop, 2018].

DATASETS. For our basket completion experiments we use the following three datasets, previously describe in section 2.2.2:

- **Amazon Baby Registries** For the purposes of comparison with [Gartrell et al., 2016], we perform two experiments. The first experiment is conducted using the diaper category, which contains 100 products and approximately 10k baskets, composed of 2.4 items per basket on average. The second experiment is performed on the concatenation of the diaper, apparel, and feeding categories (sometimes noted here D.A.F for Diaper+Apparel+Feedings), which contains 300 products and approximately 17k baskets, composed of 2.6 items per basket on average. Let recall that since the categories are disjoint, no basket containing diaper products will be observed containing apparel products, for example. This concatenation of disjoint categories may present difficulties for classic matrix factorization models [Gartrell et al., 2016], which may prevent these models from learning a good embedding of items.

- **Belgian Retail Supermarket** the entire dataset was used. AC could not be trained on this dataset because AC does not scale with large item catalogs.

- **UK retail** the entire dataset was used. Because of the presence of very large baskets, low-rank DPP requires a very large number of latent factors leading to somewhat poor results for this model. This is not an issue for our model, due to the item bias that is captured in a separate matrix. However, for purposes of comparison, we have removed all baskets containing more than 100 items, but the low-Rank DPP still requires 100 latent factors to model these baskets. AC could not be trained on this dataset because AC does not scale with large item catalogs.

- **Instacart** is the only public dataset that we know of, that contains ordered sequences of purchase. Because of memory issues, we only used the "train" dataset, removed items that appeared less than 15 times and basket of size lower than 3.

We tested both the accuracy of our models and the ability of our Multi-Task DPP algorithm to capture directed completion rules. For accuracy we used Amazon, Belgian Retail Supermarket and UK retail datasets and compared all previously introduced algorithms except the RNN model since it requires ordered sequences which is not the case with these datasets. The Instacart dataset was then used to validate our hypothesis on capturing directed completion rules by comparing the performance of Low-Rank DPP, RNN and Multi-Task DPP on various experimental protocols.

4.4.1 Accuracy

For model evaluation we use a random split of 70% of the data for training, and 30% for testing. For each basket in the test set we remove one item at random. We then evaluate the model prediction according to the predicted score of this removed item using the metrics described below.

METRICS. To evaluate the performance of each model we compute the Mean Percentile Rank (equation 2.9) and precision@k (equation 2.10) for $k = 5, 10$ and 20.

RESULTS. Looking at Table 4.1 we see that classic collaborative filtering models sometimes have difficulty providing good recommendations in this basket-completion setting. Perhaps more surprising, but already described in [Gartrell et al., 2017], is that except for the Belgian dataset, PF provides MPR performance that is approximately equivalent to a random model. For the Amazon diaper dataset this poor performance may be a result of the small size of each basket (around 2.4 items per basket on average), thus each "user" is in a cold start situation, and it is difficult to provide good predictions. Poor performance on the diaper+apparel+feedings Amazon's dataset may result from the fact that, apart from the small basket size of 2.61 items on average, this dataset is composed of three disjoint categories. These disjoint categories can break the low-rank assumption for matrix factorization-based models, as discussed in [Gartrell et al., 2016]. This issue is somewhat mitigated by FM, due to the integration of an item bias into the model. This item bias allows the model to capture item popularity and thus provide acceptable performance in some cases.
Low Rank DPP vs Multi-Task DPP. We now turn to a performance comparison between our primary baseline, the low-rank DPP model, and our model, where the relative improvements are highlighted in Table 4.2. We see that our approaches provide a substantial increase in performance for both Amazon datasets, with improvements between 10% and 70%. The first reason is that unlike the low-rank DPP, which models the probability that a set of items will be bought together, our approach directly models the basket completion task. In our MULTI-TASK DPP model, the extra dimensions allow the model to capture the correlation between each item in the basket and the target item, as well as the global coherence of the set.

Regarding the three-category Amazon dataset, a good model should not be impacted by the fact the all the three categories are disjoint. Thus the precision@K scores should be approximately the same for both the single category and three-category datasets, since we observed similar performance for each category independantly. Since the MPR is 78% for one category, the MPR on the three-category dataset should be around 93%, since on average for each category the target item is in 22nd position over 100 items. Therefore, the target item should be in 22nd position over the 300 items, resulting in a MPR of $1 - 22/300 = 93\%$. Our models come close to those numbers, but still exhibit some small degradation for the three-category dataset. Finally, we note that for this dataset, a model that samples an item at random from the right category would have a precision @20 of 20%, since there are 100 items per category. The low-rank DPP model provides close to this level of performance. Taken together, these observations indicate that our model is robust to the disjoint category problem, and explains the 70% relative improvement we see for our model on the precision@20 metric. On the UK retail dataset the improvements of our algorithm are still substantial for precision@K,

model	r	MPR	Prec.@5	Prec.@10	Prec.@20
Associative Classifier	-	-	16.66	16.66	16.66
Poisson Factorization⋆	40	50.30	4.78	10.03	19.90
Factorization Machines	60	67.92	24.01	32.62	46.25
Low Rank DPP⋆	30	71.65	25.48	35.80	49.98
Bayesian Low Rank DPP⋆	30	72.38	26.31	36.21	51.51
Logistic DPP	50	71.08	23.7	34.01	48.44
Multi-Task DPP no bias	50	77.5	32.7	45.77	61.00
Multi-Task DPP	50	**78.41**	**34.73**	**47.42**	**62.58**

(a) Amazon (diaper) dataset.

model	r	MPR	Prec.@5	Prec.@10	Prec.@20
Associative Classifier	-	-	4.16	4.16	4.16
Poisson Factorization	40	51.36	4.16	5.88	9.08
Factorization Machines	5	65.21	10.62	16.71	24.20
Low Rank DPP	30	70.10	13.10	18.59	26.92
Bayesian Low Rank DPP	30	70.55	13.59	19.51	27.83
Logistic DPP	60	69.61	12.65	19.8	27.86
Multi-Task DPP no bias	60	88.77	18.33	28.00	43.57
Multi-Task DPP	60	**89.80**	**20.53**	**30.86**	**45.79**

(b) Amazon (diaper+apparel+feedings) dataset.

model	r	MPR	Prec.@5	Prec.@10	Prec.@20
Poisson Factorization	40	87.02	21.46	23.06	23.90
Factorization Machines	10	65.08	20.85	21.10	21.37
Low Rank DPP	76	88.52	21.48	23.29	25.19
Bayesian Low Rank DPP	76	89.08	21.43	23.10	25.12
Logistic DPP	75	87.35	21.17	23.11	25.77
Multi-Task DPP no bias	75	87.42	21.02	23.35	25.13
Multi-Task DPP	75	87.72	21.46	23.37	25.57

(c) Belgian Retail Supermarket dataset.

model	r	MPR	Prec.@5	Prec.@10	Prec.@20
Poisson Factorization	100	73.12	1.77	2.31	3.01
Factorization Machines	5	56.91	0.47	0.83	1.50
Low Rank DPP	100	**82.74**	3.07	4.75	7.60
Bayesian Low Rank DPP	100	61.31	1.07	1.91	3.25
Logistic DPP	100	75.23	3.18	4.99	7.83
Multi-Task DPP no bias	100	77.67	3.82	5.98	9.11
Multi-Task DPP	100	78.25	**4.00**	**6.20**	**9.40**

(d) UK retail dataset.

Table 4.1: Performance of all models on all datasets. Best results within each dataset are in bold. r denotes the number of latent factors.

dataset	MPR	Prec.@5	Prec.@10	Prec.@20
Amazon (diaper)	**9.43%**	**36.28%**	**32.47%**	**25.2%**
Amazon (D.A.F)	**28.11%**	**56.71%**	**66.01%**	**70.11%**
Belgian Retail Supermarket	−0.9%	−0.1%	0.34%	1.52%
UK Retail	−5.43%	**30.29%**	**30.53%**	**23.68%**

Table 4.2: Improvement of Multi-Task DPP over Low Rank DPP performances in percentage.

with relative improvement between 20% and 30% (MPR is down by 5%). We also observe the same decrease in MPR for our Logistic DPP model, but precision@K similar to the low-rank DPP. Finally, on the Belgian Retail dataset we see that all models provide similar performance. For this dataset, baskets come from an offline supermarket, where it is possible that customers commonly purchased similar products at specific frequencies. Consequently it may be easy to capture frequent associations between purchased items, but very difficult to discover more unsual associations, which may explain why all models provide approximately the same performance.

Logistic DPP vs Multi-Task DPP. To better understand the incremental performance of our model, we focus on the results of the Logistic DPP and the Multi-Task DPP models. We see that the single-task model does not improve over the low-rank DPP on average, indicating that the logistic component of the model does not contribute to improved performance. However, we argue that this formulation may still be valuable in other classification applications, such as those with explicit negative feedback. For the Multi-Task DPP model, we see that the version of this model without bias is responsible for almost all of the performance improvement. Some additional lift is obtained when capturing the item popularity bias in a separate matrix. Since most of the gain comes from the multi-task kernel, one may ask if we could use the multi-task kernel without the logistic function and obtain similar results. We believe that this is not the case for two reasons. First, since we are clearly in a classification setting, it is more appropriate to use a logistic model that is directly tailored for such applications. Second, without the logistic function, each slice of the tensor should define a probability distribution, meaning that the probability of purchasing an additional product should sum to one over all possible baskets. However, we could add an arbitrarly bad product that no one would ever buy, resulting in a probability of 0 for buying that item in any basket, which would break the distributional assumption.

model	protocol	MPR	Prec.@5	Prec.@10	Prec.@20
Low Rank DPP	(1)	76.46	7.37	8.07	9.23
Multi-Task DPP	(1)	80.46	4.62	7.23	10.51
Low Rank DPP	(2)	61.16	7.49	8.05	8.8
RNN	(2)	73.31	1.08	1.99	3.2
Multi-Task DPP	(2)	**89.99**	**7.99**	**14.34**	**20.16**
Multi-Task DPP	(3)	80.65	5.23	6.05	9.72

Table 4.3: Performance of the models on Instacart dataset. All models used 80 latent factors.

4.4.2 Capturing directed completion rules

In order to validate the capacity of our model to capture directed completion rules we performed three experimental protocols on the Instacart dataset, which is the only one to have ordered sequences. Each protocol vary in the way we removed the item to predict from the basket:

1. As for previous experiements, we removed one item at random. For Low-Rank DPP this is done only in the test set, for Multi-Task DPP both for training and test sets.

2. We removed the last added item to the basket. For Low-Rank DPP this is done only in the test set, for Multi-Task DPP both for training and test sets. Since we considered ordered sequence, we also tested the RNN model here.

3. We removed one item at random in the training set, and the last added item in the test set. Here we only evaluate the performance of our Multi-Task DPP.

RESULTS. First, looking at table 4.3 and comparing Multi-Task DPP results of protocols (2) and (3), we see that the algorithm is much more precise to predict the last added item in the test set when training is also done by removing the last added item. This support the idea that the order in which the items are added in the basket plays an important role, otherwise both protocols would give similar results. Second, comparing results of protocols (1) and (2), we see that Multi-Task DPP performance is lower when trained to predict a randomly removed item than when trained to predict the last added item whereas it is the other way around for Low-Rank DPP but by a smaller margin. This further support the fact that Low-Rank DPP, although well suited to compute co-occurence probabilities, misses directed completion rules. Finally, we see that, quite surprisingly, RNN model does not obtain a good performance. This poor performance may come from the fact that the basket length are too small for the RNN to learn correctly.

4.5 Related Work

The topics of DPPs, basket completion, and diversity have received significant attention in the past several years.

4.5.1 Basket Completion

The problem of basket completion, introduce in subsection 2.1.2 is a particular problem in session-based recommender systems, where the basket (also called the cart) refers to all of the products that a user has saved and is about to purchase. The goal is to recommend one or several items that the user could add to the basket.

Theoretically this problem could be addressed using MDPs, where one would look for the next item to show based on the items the user has previously interacted with ([Shani et al., 2005]) but those techniques are very difficult to scale with very large items catalogs, since the state space is large and learning transition probabilities and rewards will be infeasible. One solution for addressing the basket completion problem would be to use the classic collaborative filtering approach that considers each basket as a new user. However such an approach has many drawbacks. From a computational and storage point of view, this approach would hugely increase the size and the sparsity of the user-item interaction matrix, making matrix completion very difficult. Other limitations include short dependencies and the global coherence of the set of items (basket) required for basket completion, which are not taken into account with a conventional collaborative filtering approach. Indeed, the first choice of the user must condition the rest of the recommendations, which may not be possible if the recommendation are only updated in an offline fashion (for instance every night). Additionally, item associations may be directed links: if a user is buying a couch, she may add a pillow to her cart, but the other direction is much less likely. Such directed interactions are not tackled by state-of-the-art collaborative filtering approaches like matrix factorization. Since the similarity computation using latent factors in a matrix factorization model is commutative, there is nothing that prevents the system from recommending a couch to someone who is about to purchase a pillow. Furthermore, if one pillow is frequently purchased with one couch and just as frequently with another couch, matrix factorization is likely to recommend a second couch to someone about to purchase the first one, which is even more uncommon.

Associative classification is a simple method that uses association rules to solve basket completion. More advance methods based on DPPs have shown very good accuracy and improve diversity in the recommendation. Recently deep learning techniques based on recurrent neural networks have been developed for this task.

With Associative Classification

To solve the basket completion problem, Associative Classifiers have long been the state-of-the-art [Agrawal et al., 1993, Hipp et al., 2000, Liu et al., 1998], despite requiring very heavy computational load for training, and manual tuning for key parameter choices such as lift and confidence thresholds. The objective of the Associative Classifier is to find frequent itemsets in order to extract association rules, for instance *"Propose item A to anyone that has B and C in her basket"*. In our application this mean that many different users have purchased the same set of products within one session. However, the basket completion problem is a directed problem, if the itemset (A, B) (where A and B can represent one or several items) appeared frequently, it does not necessarily means that one must recommend B to anyone who bought A. With associative classification this problem is addressed by computing the confidence of a rule ([Hahsler et al., 2005]):

Definition 8 (Confidence rule).

$$conf(A \to B) = \frac{supp(A \cup B)}{supp(A)} \tag{4.45}$$

where $supp(A)$ is the support of A, that is, the number of times product A has been observed in a basket. The idea is to weight the support of the item set (A, B) based on the support of A. Indeed, the associations (A, B) may have been observed frequently only because A is one of the most popular items. Let us illustrate this by the "pillow-couch" problem shown in table 4.4. The set (pillow, couch) appears quite often, and one may recommend one of these items if someone has purchased the other item. However $conf$(pillow $\to$ couch) $= 2575/12432 \approx 0.21$, and $conf$(couch $\to$ pillow) $= 2575/3021 \approx 0.85$, which shows that one may recommend a pillow to a couch buyer but not the other way around. This confidence score can be seen as an estimate of the probability of (observing the association) $A \cup B$, knowing A ([Hipp et al., 2000]). In practice one must fix a threshold in order to select only relevant rules. A good association rule must satisfy both a minimum support and confidence score at the same time. However in order to not reduce to most popular associations and loose the interest of users, the selection must be made among medium to low support thresholds, and too many associations may be returned by this selection strategy. Among other possibilities, a common solution is to filter out association rules based on an additional relevancy measure. [Brin et al., 1997] proposed a measure, called *lift*, that weights the support of the set union by the support of each subset independantly:

Definition 9 (Lift rule).

$$lift(A \to B) = \frac{supp(A \cup B)}{supp(A)supp(B)} \tag{4.46}$$

Thus high *lift* indicates strong association.

Association Classification has several limits. The first one is that one must manually choose thresholds for minimum support, confidence, and lift before outputing the recommendation. The second is the high computational load of the system, which increases with the number of transactions, the length of the average transaction,

basket	number of occurrences
pillow	12 432
lamp	8 659
pillow, lamp	4 374
couch	3 021
pillow, couch	2 575
⋮	⋮
table	251
wardrobe	121

Table 4.4: Example of a several basket observations

and the number of recommendations to produce (see [Tan et al., 2005]). Because of these constraints, it is in practice impossible to apply Association Classification to large item catalogs. To overcome these issues, some work uses Determinantal Point Processes to model and learn the distribution of items co-occurences.

With Determinantal Point Processes

Several previous works proposed the use of DPPs and leverage their repulsive behaviour to model the probability of observing a set of items, and then use this model to produce basket-completion recommendation. In contrast to this work, we directly model the probability of purchasing a particular item according to the products in the basket. [Gillenwater et al., 2014] proposed an Expectation-Maximization algorithm to learn a full-rank kernel while [Mariet and Sra, 2015] relied on a fixed-point algorithm. However when dealing with very large item catalogs learning a full-rank kernel can be infeasible. To resolve this issue, [Gartrell et al., 2017] (respectively [Gartrell et al., 2016]) proposed a (respectively bayesian) low-rank DPP. They assume that the kernel K can be factorized into $K = VV^T$ where the number of columns of V, that is the rank of the kernel, is much smaller than the catalog size, which enables faster learning and predication algorithms that can scale to larger catalog. Furthermore, low-rank DPPs often have better predictive performance than full-rank DPP due to the use of an appropriate regularization.

With Recurrent Neural Networks

Recently, deep learning techniques have been investigated to provide solutions for session-based recommender systems. The problem can be seen as predicting the next item, and has already been investigated in natural language processing to produce word embeddings by learning to predict the next word using deep learning ([Mikolov et al., 2013a, Mikolov et al., 2013b, Chen, 2017]). In both applications (words or recommender systems) each item is map to its unique position in a dictio-

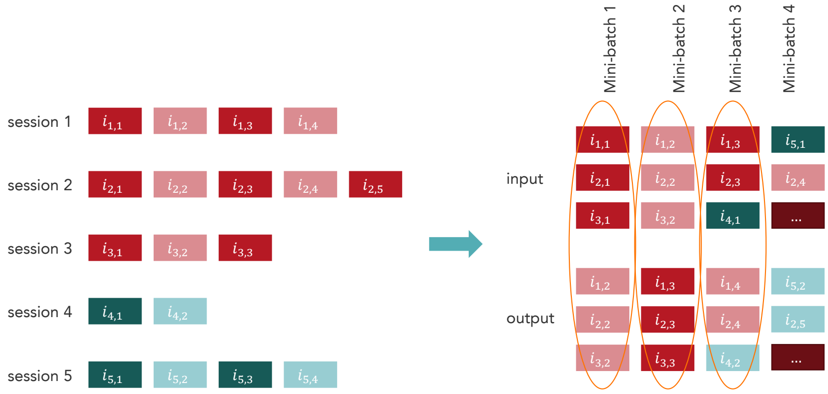


Figure 4.2: Session-parallel mini-batch creation.

nary of size the vocabulary or catalog size. [Hidasi et al., 2015] proposed a Recurrent Neural Network solution to tackle the session-based recommendation problem. For such models, the training set usually consists of a unique very long sentence by concatenating all sentences and mini-batch learning is applied by extracting a portion of the text with a sliding window. When applied to session-based recommendation, [Hidasi et al., 2015] observed poor performance with this approach. This problem comes from the fact that sessions can have very different length that can be very short or very long and significant amounts of information are lost when breaking down sessions. This issue can be solved with an appropriate mini-batching logic: first sessions are ordered, then the first mini-batch is created by selecting the first event of the first X sessions with the second event as an output. The second mini-batch is then formed from the second and third events and so on. When a session comes to an end the next available session is appended to it. See Figure 4.2. Several variations of this appraoch have been developed to further improved RNNs for the basket-completion task, by adding multiple types of features like text and image ([Hidasi et al., 2016], with explicit context ([Twardowski, 2016]) or by taking some temporal dynamics into account ([Smirnova and Vasile, 2017]).

4.5.2 Determinantal Point Processes

In addition to the previously discussed work, DPPs have been used for natural language processing in order to discover diverse threads of documents [Gillenwater et al., 2012], and to enhance diversity in recommender systems [Foulds and Görür, 2013, Chen et al., 2017]. Unlike our application where we learn the kernels, in these applications the kernel is provided using previously obtained latent factors, for instance using tf-idf for [Gillenwater et al., 2012]. These latent factors are scaled by a relevance score learned in a more conventional fashion. For example, these relevance scores may represent the predicted ratings of a particular user on each item/document, or the similarity between the text in a document and the user query. Ultimately, these applications involve sampling from the DPP specified by this kernel, where the kernel parameters trade off between relevance and diversity. However, sampling from such a DPP efficiently is difficult, and this has lead to work on different sampling techniques. Ref. [Han et al., 2017, Gillenwater et al., 2014, Gautier et al., 2017] rely on MCMC sampling, while [Chen et al., 2017] proposes a greedy solution based on Cholesky decomposition.

Several algorithms have been proposed for learning the DPP kernel matrix. Ref. [Gillenwater et al., 2014] uses an expectation-maximization (EM) algorithm to learn a non-parametric form of the DPP kernel matrix. Ref. [Mariet and Sra, 2015] proposed a fixed-point algorithm called Picard iteration, which proved to be much faster than EM, but still slower than [Gartrell et al., 2017]. Bayesian learning methods have also been proposed to learn the DPP kernel [Gartrell et al., 2016, Affandi et al., 2014].

4.5.3 Diversity

Improving diversity in recommender systems has also been studied without the use of DPPs, including, among other work, in [Christoffel et al., 2015, Puthiya Parambath et al., 2016, Teo et al., 2016, Vargas and Castells, 2014, Paudel et al., 2017]. For instance, [Christoffel et al., 2015] relies on random walk technics to enhance diversity. In [Puthiya Parambath et al., 2016], the authors propose trading off between the relevance of the recommendation and diversity by introducing a coverage function to force the algorithm to produce recommendations that cover different centers of interests of each user. Ref. [Paudel et al., 2017] proposes extensions of logistic matrix factorization for two-class collaborative filtering. One of the methods presented in this paper learns two sets of latent item factors, one for each class, that represent positive and negative feedback, and show the effectiveness of this approach in producing recommendations that better cover the catalog while maintaining accurate recommendations. Finally, the authors of [Vargas and Castells, 2014] propose transforming the problem of recommending items to users into recommending users to items. They introduce a modification of nearest-neighbor methods, and a probabilistic model that allows isolation of the popularity bias and favors less popular items.

4.6 Industrial learnings

For a long time, associative classifiers have been used at FIFTY-FIVE for basket completion problems. However, large catalog sizes and large number of baskets to process always made associative classifiers difficult to use. Other limits for industrial applications include the choice of the different thresholds which blur the results and the infeasibility of performing real-time predictions. Indeed with associative classifers, the associations rules have to be computed offline and recommendation are generated thanks to a look up table, which is impossible in practice. Low-rank DPP and the solution of [Gartrell et al., 2017] then became the new production solution and subsequently Logistic DPP because of its better accuracy. DPP solutions have the advantage in terms of computation time and prediction time. Indeed it is easy to generate a recommendation for a basket that has never been previously seen.

4.7 Conclusion

In this section we have proposed an extension of the DPP model that leverages ideas from tensor factorization. While our model can be applied to a number of machine learning problems, we focus on the problem of basket completion. We have shown through experiments on several datasets that our model provides signficant improvements in predictive quality compared to a number of competing state-of-the-art approaches. Besides, we showed that for basket completion, there may be a significant signal in the order in which items are added to the basket, and, unlike state-of-the-art DPPs solutions, our Multi-Task DPP appropriately captures this signal. In future work we plan to investigate other applications of our model, such as user conversion prediction, attribution, and adversarial settings in games. We also plan to investigate better negative sampling methods for positive-only and unlabelled data. Finally, we also plan to investigate other types of loss functions, such as hinge loss, and other types of link functions for DPPs, such as the Poisson function, to tailor DPPs for regression problems. We believe that this work will allow us to customize DPPs so that they are suitable for many additional applications.

Fighting Boredom in Recommender Systems with Linear Reinforcement Learning

The goal of A/B testing is to discover the best option between two alternative strategies. This implicitly assumes that the best option is indeed to exclusively execute either A or B. We argue that this assumption, which is common in many recommender systems, is rarely verified in practice and that alternating between different solutions (possibly more than two solutions) with the right frequency may significantly improve the performance with respect to sticking to the best *fixed* one. This performance improvement is rooted into the dynamics of the recommendation process. A user may get bored if shown too many movies of the same genre in a short time, while she may get interested in it again, if enough time passed since the last time she watched a movie of that genre. While fatigue variables are commonly integrated in recommender systems, their impact on the recommendation is often limited and they do not effectively take into consideration how recommendations themselves influence the preferences of users. In this section, we show that it is critical to estimate the dependency between the rewards (e.g., ratings, clicks, time usage, ...) and the past recommendations produced by the system in order to accurately make recommendation that will interest the user on the long term. We first cast the problem as a Markov Decision Process, where the rewards are a linear function of the recent history of actions, and we show that a policy taking into consideration how recommendations influence the rewards in the long run, may achieve a much better performance with respect to either fixed-action or even greedy policies (i.e., choosing the best action at each time step). We then introduce an extension of the UCRL algorithm (LinUCRL) for which we derive a regret bound that is independent of the number of states. Finally, we experimentally validate the model assumptions and the algorithm in a number of realistic scenarios.

5.1 Introduction

Consider a movie recommendation problem, where the recommender system selects the genre of movie to suggest to a user and then picks a movie from this genre. A very basic strategy is to evaluate user's preferences on genres and then keep recommending movies of the preferred genre. While this strategy is sensible in the short term, it completely overlooks the *dynamics* of the user's preferences due to the recommendation process itself. In fact, the user would rapidly get bored of the proposed genre and reduce the ratings accordingly. Notice that this is due to the recommendation strategy and not because of an actual change in the preferences of the user, as she would still like the genre as before had it not been proposed so often recently. Despite being clearly suboptimal and basic, the existence of an optimal *fixed* strategy is often assumed in recommender systems using, e.g., matrix factorization to estimate users' ratings and the best (fixed) item. Even learning strategies based on the multi-armed bandit (MAB) model assume constant rewards and pursue the objective of selecting the (fixed) optimal arm as often as possible. Similarly, in *A/B* testing, we implicitly assume that once the best option (either *A* or *B*) is found, it should be constantly executed. An alternative approach is to estimate the *state* of the user (e.g., her level of boredom) as a function of the movies recently watched and determine how her preferences are affected by that. Thus we could design a *contextual* strategy that recommends the best genre *depending* on the actual state of the user. This strategy could be effectively learned using a linear bandit model, where the reward of an arm is a linear combination between the context and some unknown vector. While this could partially address the previous issue, we argue that in practice it may not be satisfactory. In fact, the state of the user directly depends on the sequence of movies recommended in the recent past and a recommendation strategy should take into consideration this dynamics to "drive" the user's state in the most favorable condition to gain as much reward in the long term, instead of following greedy strategy that selects the best "instantaneous" action at each step. For instance, consider a user with preferences *1) action*, *2) drama*, *3) comedy*. After showing a few *action* and *drama* movies, the user may get bored. At this point a greedy contextual strategy would successfully move to recommending *comedy*, but as soon as it estimates that *action* or *drama* are better again (i.e., since they have not been watched for a while, their reward would go back to a higher value), it would immediately switch back to them. On the other hand, a more farsighted strategy may prefer to stick to *comedy* for a little longer to increase the preference of the user for *action* to its higher level, so as to fully exploit its potential. In this section, we propose to use a reinforcement learning model to fully capture this dynamical structure, where the reward (e.g., the average rating of a genre) depends on a state that summarizes the effect of the movies recently watched by the user on her preferences. We introduce a novel learning algorithm that effectively trades off exploration and exploitation and we derive theoretical guarantees for its performance. Finally we validate our model and our theoretical

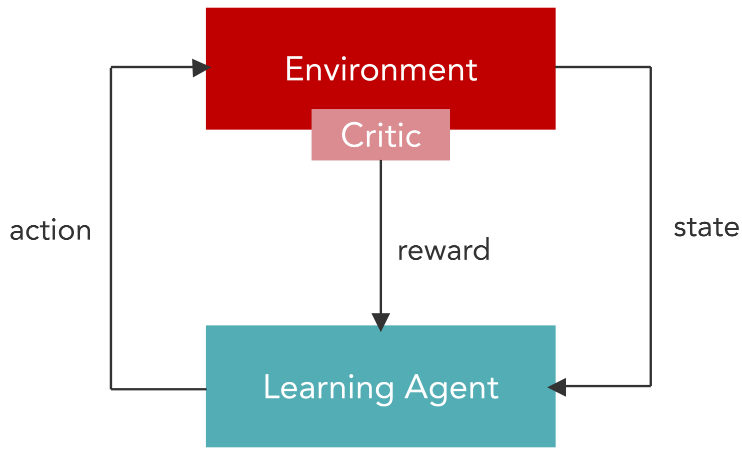


Figure 5.1: The Agent-Environment Interaction Protocol

findings in a number of environments either synthetic or built on real data.

5.2 Background

5.2.1 Linear bandit

We already discussed about multi-arm bandit and linear bandit in section 2.1.3. We will now introduce more theoretical background on linear bandit that will be leveraged in this chapter. Let recall that in linear bandit when arm a is pulled, the learner received a stochastic reward r_a which is a linear function of a vector $x_a \in \mathbb{R}^d$ characterizing arm a, i.e, $r_a =< \theta^*, x_a > +\eta$, where $\theta^* \in \mathbb{R}^d$ is an unknown parameter.

Following the principle of optimism in face of uncertainty leads to a solution called OFUL (Optimism in the Face of Uncertainty Linear bandit algorithm) the algorithm will select the next arm according to a trade-off between the prediction of the linear regression and a confidence interval on the prediction. Let X_t be the matrix whose rows are $x_{a_1}, \cdots, x_{a_t}$ with a_τ is the arm pulled at time τ, $Y_t = [r_{a_1}, \cdots, r_{a_t}]$, and let $\hat{\theta}_t$ be the ℓ^2-regularized least-squares estimate of θ^* at time t with regularization parameter λ, then

$$\hat{\theta}_t = (X_t^T X_t + \lambda I)^{-1} X_t^T Y_t \tag{5.1}$$

Then one can construct a confidence ellipsoid around $\hat{\theta}_t$ where θ^* lies with high probability:

Theorem 7 (simplify Theorem 2 of [Abbasi-yadkori et al., 2011]). *Assume that* $||\theta^*||_2 \leq S$ *and* $||x_a||_2 \leq L$ *for all arm* a*, then, for any* $\delta > 0$*, with probability at least* $1-\delta$*, for all* $t > 0$ θ^* *lies in the set*

$$C_t = \left\{ \theta \in \mathbb{R}^d : ||\hat{\theta}_t - \theta||_{V_t} \leq R\sqrt{d \log\left(\frac{1+tL^2/\lambda}{\delta}\right)} + \lambda^{1/2} S \right\}$$

with $V_t = X_t^T X_t + \lambda I$

Finally the system pulls the arm at time t according to the following rule

$$(a, \tilde{\theta}_t) = \arg\max_{(a,\theta)\in\mathcal{A}\times C_{t-1}} < \theta, x > \tag{5.2}$$

5.2.2 Reinforcement Learning

In reinforcement learning, an agent interacts with the environment in order to maximize long term performance (see figure 5.1). To do that the agent selects an action to execute in each environment state in order to simultaneously learn the environment - the transition between states and the value of each state - and gather as much reward as possible. In such context, the agent choices have a direct impact on future observations that is future states. The agent is given a set of possible actions $\mathcal{A}$, a set of possible states $\mathcal{X}$ of cardinality N but does not know anything about the dynamics - that is the transition probabilities between states and the reward available in each state. By moving inside this environment the agent collects information to estimate this dynamic. This environment is then defined as an MDP $M = \langle \mathcal{X}, \mathcal{A}, p, r\rangle$, with $p : \mathcal{X} \times \mathcal{X} \times \mathcal{A} \to \mathbb{R}^3$ which value $p(y, x, a)$ holds the transition probability of moving to state y from state x when playing action a, and $r : \mathcal{X} \times \mathcal{A} \to \mathbb{R}$ is the (stochastic) reward obtained when playing a given action in a given state. A greedy approach can consist in exploring the environment randomly during a fixed amount of time and then decide what policy $\pi : \mathcal{X} \to \mathcal{A}$ to run - what action to execute in each state - given gathered knowledge. In order to link collected knowledge with policy and dynamic, some definitions must be introduced.

Definition 10 (State value function in infinite time horizon with discount).

$$V^\pi(x) = \mathbb{E}[\sum_{t=0}^{\infty} \gamma^t r(x_t, \pi(x_t))|x_0 = x; \pi]$$

where $0 \leq \gamma \leq 1$ is a discount factor, x_t the state at time t and π the policy, that is the function that output which action to perform in state x.

Algorithm 9 The Value Iteration algorithm.

Input: reward function r and transition probabilities p.
Initialization: Let $V_0 \in \mathbb{R}^N$
while not converge **do**
  $V_{k+1} = \mathcal{T} V_k$
end while
Compute greedy policy after K updates: $\pi_K(x) = \arg\max_{a \in \mathcal{A}}[r(x,a) + \gamma \sum_y p(y|x,a) V_K(y)]$

Definition 11 (Optimal policy and optimal value function).

$$\pi^* = \arg\max_{\pi \in \Pi} V^\pi$$

in all the states $x \in \mathcal{X}$, where Π is some policy set of interest. The corresponding value function is the optimal value function $V^* = V^\pi$.

Based on definition 10 we see that the value function measures the cumulated reward gathered by a given policy. The objective is then to find the optimal policy (or equivalently the optimal value function) that is the policy that gathers the maximal amount of rewards. To compute the optimal value function a solution consists in using the Bellman operator.

Definition 12 ((Optimal) Bellman Operator). *For any $W \in \mathbb{R}^N$, the* ***Bellman operator*** *$\mathcal{T}^\pi : \mathbb{R}^N \to \mathbb{R}^N$ is defined as*

$$\mathcal{T}^\pi W(x) = r(x, \pi(x)) + \gamma \sum_y p(y|x, \pi(x)) W(y), \tag{5.3}$$

and the ***optimal Bellman operator*** *is defined as*

$$\mathcal{T} W(x) = \max_{a \in \mathcal{A}} [r(x,a) + \gamma \sum_y p(y|x,a) W(y)] \tag{5.4}$$

Proposition 8 ([Bertsekas and Tsitsiklis, 1996]). *Under several common assumptions, the optimal value function is bounded, i.e $||V^*||_\infty < \infty$ and it is the unique fixed point of the* ***optimal*** *Bellman operator $\mathcal{T}$. Furthermore, we have that $\forall W \in \mathbb{R}^N, V^* = \lim_{k \to \infty} (\mathcal{T})^k W$.*

It can be shown that the (optimal) Bellman operator is a contraction, thus one can use the fixed-point theorem to estimate the optimal value function and the associated policy. The protocol to achieve this goal is called Value Iteration, see algorithm 9.

In practice Value Iteration is run until a certain difference between two sucessive values of the value function is reached. The proposition 9 gives guarantee on the policy once such a difference is reached.

Proposition 9 ([Puterman, 1994]). *If value iteration is stopped at iteration n when $\max_s(u_{i+1}(s) - u_i(s)) - \min_s(u_{i+1}(s) - u_i(s)) \leq \varepsilon/2$, then the greedy policy $\pi_n(s) = \max_{a\in[K]}[r(s,a) + u_n(f(s,a))]$ is ε optimal (i.e., $\eta^\pi \geq \eta^* - \varepsilon$).*

After some exploration of the states, an estimation of the reward function r and transitions p can be computed. One can then applied the Value Iteration algorithm to decide how to behave in the future. However such an exploration can be really unefficient and take a very long time. An alternative can consist in applying an ε-greedy policy that is acting at random $\varepsilon\%$ of the time and accordly to the computed policy otherwise. The policy can then be recomputed using Value Iteration from time to time. More efficient than the previous strategy, this variation can still be very suboptimal. To overcome this issue, [Jaksch et al., 2010] leveraged a classic idea in reinforcement learning, optimism in face of uncertainty, to provide a more efficient exploration-exploitation algorithm. Indeed instead of acting accordly to a policy that has been evaluated using few feedbacks, their algorithm called UCRL, compute confidence intervals around rewards and transitions probabilities. This defined a set of plausible MDPs, each with an associated mean reward when playing the associated policy for an infinite time. The UCRL algorithm chooses the optimistic environment to select the new policy that is the MDP that obtain the highest mean reward. A new policy is then computed whenever a confidence interval is halved.

5.3 Problem Formulation

We consider a finite set of actions $a \in \{1, \ldots, K\} = [K]$. We define the state s_t at time t as the history of the last w actions, i.e., $s_t = (a_{t-1}, \cdots, a_{t-w})$, where for $w = 0$ the state reduces to the empty history. As described in the introduction, and unlike the standard MAB setting, we expect the reward of an action a to depend on how often a has been recently selected (e.g., a user may get bored the more a recommendation is used). More formally, we introduce the recency function $\rho(s_t, a) = \sum_{\tau=1}^{w} \mathbb{1}\{a_{t-\tau} = a\}/\tau$, where the recency of an action decreases as $1/\tau$ as it is selected τ steps ago. The choice of this $1/\tau$ decrease is quite arbitrary and must be validated with the data. One may choose other form of decrease to define how fast users forget about the past, like for instance $1/\tau^2, 1/\sqrt{\tau}$, or $1/\exp(\tau)$. If an action is often selected the recency is large, and the more recent the larger, while it decreases if not selected for a while. We define the (expected) reward function associated to an action a in state s as[1]

$$r(s_t, a) = \sum_{j=0}^{d} \theta^*_{a,j} \rho(s_t, a)^j = x_{s,a}^\mathsf{T} \theta^*_a, \tag{5.5}$$

where $x_{s,a} = [1,\ \rho(s,a),\ \cdots,\ \rho(s,a)^d] \in \mathbb{R}^{d+1}$ is the context vector associated to action a in state s and $\theta^*_a \in \mathbb{R}^{d+1}$ is an unknown vector. Notice that for $d = 0$ or $w = 0$, this model reduces to the standard MAB setting, where the first component $\theta^*_{a,0}$ represents the expected reward associated to arm a. Eq. (5.5) extends this setting by assuming that the reward of an action is the sum between the "stationary" expected reward $\theta^*_{a,0}$ and a polynomial of the recency $\rho(s_t, a)$. Such a formulation allows to capture any kind of evolution. With $d = 1$ their could be only decreasing or only increasing reward, but increasing the value of d allows to model more complexe evolution shapes. While the algorithmic solution we propose is agnostic to the definition of recency, in the next section we validate this model with real data that display some typical relationships between rewards and recency (i.e., monotonically decreasing or increasing and then decreasing) can be accurately approximated by a polynomial of small degree.

The formulation in Eq. (5.5) may suggest that this is just a specific instance of a linear bandit problem, where $x_{s_t,a}$ is the context for action a at time t and θ^*_a is the unknown vector. Nonetheless, in linear bandit the sequence of contexts $\{x_{s_t,a}\}_t$ is not affected by the actions selected over time and the optimal action at time t is $a^*_t = \arg\max_{a\in[K]} x_{s_t,a}^\mathsf{T} \theta^*_a$,[2] while in our model, $x_{s_t,a}$ actually depends on the state s_t which summarizes the history of actions selected up to w steps in the past and an optimal policy should take into account its effect on the state to maximize the long-term average reward. We thus introduce the **deterministic** Markov decision process (MDP) $M = \langle S, [K], f, r\rangle$ with state space S enumerating the possible histories of actions of length w (note that $|S| = K^w$ which can be really large), action space $[K]$, noisy reward function in Eq. (5.5), and a deterministic transition function $f : S \times [K] \to S$ that simply drops the action selected w steps ago and appends the last action to the state, such that $f([a_{t-1}, \cdots, a_{t-w}], a_t) = [a_t, \cdots, a_{t-w-1}]$. A policy $\pi : S \to [K]$ is evaluated according to its long-term average reward as

$$\eta^\pi = \lim_{n\to\infty} \mathbb{E}\left[\frac{1}{n}\sum_{t=1}^{n} r_t\right] \tag{5.6}$$

where r_t is the (random) reward associated to state s_t and action $a_t = \pi(s_t)$. The optimal policy is thus $\pi^* = \arg\max_\pi \eta^\pi$. While an explicit form for π^* cannot be obtained in general, intuitively in our model an optimal policy may select an action with suboptimal instantaneous reward (i.e., action $a_t = \pi(s_t)$ is s.t. $r(s_t, a_t) < \max_a r(s_t, a)$) so as to let other (potentially more rewarding) actions "recharge" and

[1] The actual reward observed when selecting a at s_t is $r_t = r(s_t, a) + \varepsilon$, with ε a zero-mean gaussian noise.

[2] We will refer to this strategy as "greedy" policy thereafter.

select them later on. In general, this results into a policy that alternates actions with a fixed schedule/frequency. Indeed since the number of states and actions are finite, eventually one will visit a certain state twice and choose the same action as last time that will transite to the same next state as first time since transition are deterministic. More insights about this behavior are provided in the experimental section. Let $\eta^* = \eta^{\pi^*}$ be the optimal average reward, if the parameters θ_a^* were known, we could compute the optimal policy by using value iteration where a value function $u_0 \in \mathbb{R}^S$ is iteratively updated as

$$u_{i+1}(s) = \max_{a\in[K]} \Big[r(s,a) + u_i\big(f(s,a)\big) \Big], \tag{5.7}$$

and a nearly-optimal policy is obtained after n iterations as $\pi(s) = \max_{a\in[K]}[r(s,a) + u_n(f(s,a))]$. Note that here the iteration process is ease because of the determinisc transition function. The objective of a learning algorithm is to approach the performance of the optimal policy as quickly as possible. This can be measured by the *regret* of an algorithm $\mathcal{A}lg$, which compares the reward cumulated by $\mathcal{A}lg$ over T steps to the reward obtained by the optimal policy over the same amount of time, i.e.,

$$\Delta(\mathcal{A}lg, T) = T\eta^* - \sum_{t=1}^{T} r(s_t, a_t), \tag{5.8}$$

where (s_t, a_t) is the sequence of states and actions observed and selected by $\mathcal{A}lg$ and $r(s_t, a_t)$ is computed as in Eq. 5.5.

5.4 Model Validation on Real Data

In order to test the validity of our model to actually capture non stationnary behaviour in the recommendation process choices, we use the *movielens-100k* dataset [Herlocker et al., 1999a]. We consider a recommender system that decides the genre of movie to recommend to a user. The model introduced in the previous section predicts that user's preferences are strictly related to the number of movies of the same genre a user have recently watched (e.g., she may get bored about a genre after seeing too many of such movies and then getting interested again as time goes by without watching that genre). In order to verify this intuition, we sort ratings for each user using their timestamps to produce an ordered sequence of ratings. For different genres observed more than $10,000$ times, we compute the average rating (the reward) among all users for each value of the recency function $\rho(s_t, a)$ corresponding to the different states s_t encountered in the dataset for a history window $w = 10$. The charts of Fig. 5.2 provides a first qualitative support for our model. In fact, the rating for *comedy*, *action*, and *thriller* genres is a monotonically decreasing function of the recency, hinting to the existence of a boredom-effect, so that the rating of a genre decreases with how many movies of that kind have been recently watched. On the other hand, *drama* shows a more sophisticated behavior where users "discover"

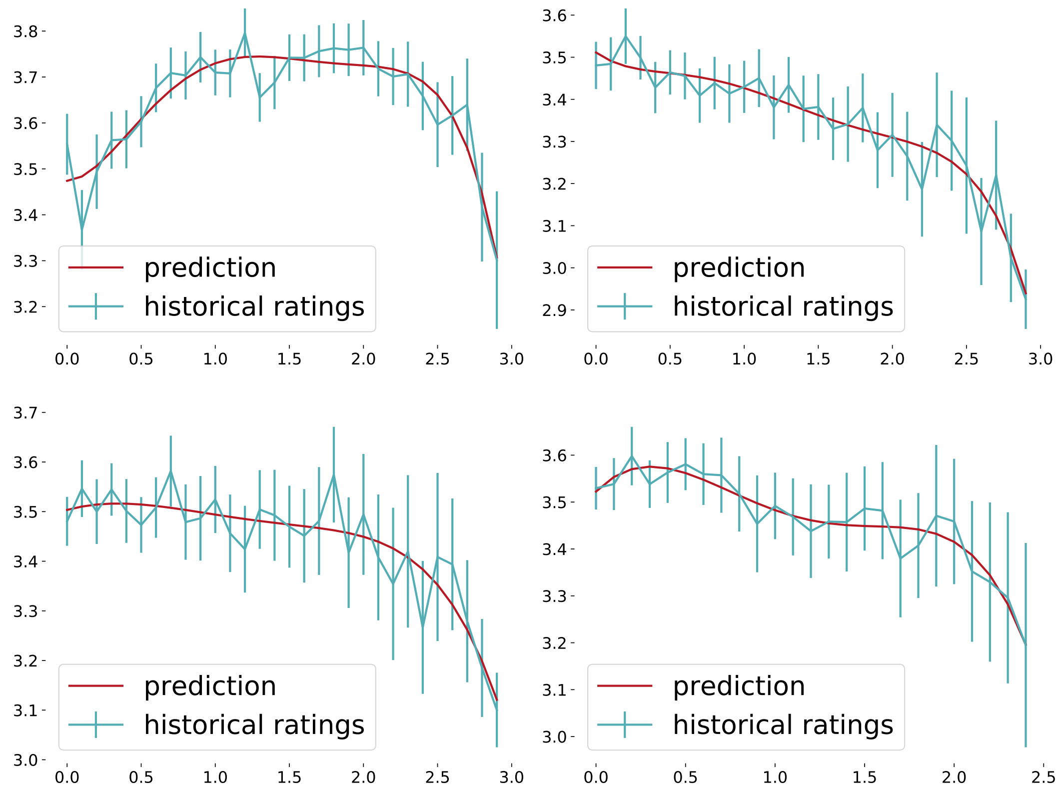


Figure 5.2: Reward (average rating) as a function of the recency for different genre of movies ($w = 10$) and predictions of our model for $d = 5$ in red. From left to right, top: *drama* and *comedy*, bottom: *action* and *thriller*. The confidence intervals are constructed based on the amount of samples available at each state s and the red curves are obtained by fitting the data with the model in Eq. 5.5.

Genre	$d=1$	$d=2$	$d=3$	$d=4$	$d=5$	$d=6$
action	0.55	0.74	0.79	0.81	0.81	0.82
comedy	0.77	0.85	0.88	0.90	0.90	0.91
drama	0.0	0.77	0.80	0.83	0.86	0.87
thriller	0.74	0.81	0.83	0.91	0.91	0.91

Table 5.1: R^2 for the different genres and values of d on *movielens-100k* and a window $w = 10$.

the genre and increase the ratings as they watch more movies, but get bored if they recently watched "too many" drama movies. This suggests that in this case there is a critical frequency at which users enjoy movies of this genre. In order to capture the multiple shapes of dependency between rating and recency for different genre, in Eq. 5.5 we defined the reward as a polynomial of $\rho(s_t, a)$ with coefficients that are specific for each action a. In Table 5.1 we report the coefficient of determination R^2 of fitting the model of Eq. 5.5 to the dataset for different values of d and different genres. The results show how our model becomes more and more accurate as we increase the complexity of the model. Fig. 5.2 shows that even a small degree ($d = 5$) actually produces accurate reward predictions, suggesting that the recency does really capture the key elements of the state s and that a relatively simple function of ρ is enough to accurately predict the rating. This result also suggests that standard approaches in recommender systems, such as matrix factorization, where the rating is contextual (as it depends on features of both users and movies/genres) but *static*, potentially ignore a critical dimension of the problem that is related to the *dynamics* of the recommendation process itself.

In practice, this genre recommender system learned using our LinUCRL algorithm can be coupled with a classic recommender system (for instance using matrix factorization) that compute user-movie affinities to form an hybrid recommender system. Indeed, the genre recommender system first decide what genre to recommend in order to keep the user interesting by diversifying the recommendation, then the second system decides exactly what movie to recommend that match the genre. In july 2018, Netflix encourage its users through a little ad (see figure 5.3) to create different profiles according to the genre of movie their are in the mood to watch, which illustrate the interest of our algorithm that directly model the genre mood of a user.

We now have showed that our model can indeed capture dynamic dependency between preference and past choices. However in practice such historical data are not available and thus must be collected while minimizing the regret 5.8. This is what we solve in the next section.

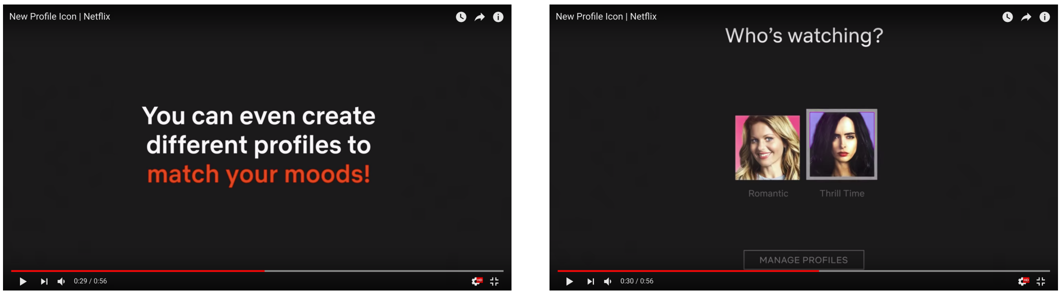


Figure 5.3: Screenshot of Netflix ad to encourage users create multiple genre profiles.

5.5 Linear Upper-Confidence bound for Reinforcement Learning

The Learning Algorithm. LinUCRL directly builds on the UCRL algorithm [Jaksch et al., 2010] and exploits the linear structure of the reward function and the deterministic and known transition function f. The core idea of LinUCRL is to construct confidence intervals on the reward function and apply the optimism-in-face-of-uncertainty principle to compute an optimistic policy. The structure of LinUCRL is illustrated in Alg. 10. Let us consider an episode k starting at time t, LinUCRL first uses the current samples collected for each action a separately to compute an estimate $\widehat{\theta}_{t,a}$ by regularized least squares, i.e.,

$$\widehat{\theta}_{t,a} = \min_{\theta} \sum_{\tau<t: a_\tau = a} \left(x_{s_\tau,a}^\mathsf{T}\theta - r_\tau\right)^2 + \lambda\|\theta\|_2^2, \tag{5.9}$$

where $x_{s_\tau,a}$ is the context vector corresponding to state s_τ and r_τ is the (noisy) reward observed at time τ. Let be $R_{a,t}$ the vector of rewards obtained up to time t when a was executed and $X_{a,t}$ the feature matrix corresponding to the contexts observed so far, then $V_{t,a} = \left(X_{t,a}^\mathsf{T} X_{t,a} + \lambda I\right) \in \mathbb{R}^{(d+1)\times(d+1)}$ is the design matrix. The closed-form solution of the estimate is $\widehat{\theta}_{t,a} = V_{t,a}^{-1} X_{t,a}^\mathsf{T} R_{t,a}$, which gives an estimated reward function $\widehat{r}_t(s,a) = x_{s,a}^\mathsf{T}\widehat{\theta}_{t,a}$. Instead of computing the optimal policy according to the estimated reward, we compute the upper-confidence bound

$$\widetilde{r}_t(s,a) = \widehat{r}_t(s,a) + c_{t,a}\|x_{s,a}\|_{V_{t,a}^{-1}}, \tag{5.10}$$

where $c_{t,a}$ is a scaling factor whose explicit form is provided in Prop. 11. Since the transition function f is deterministic and known, we then simply apply the value iteration scheme in Eq. 5.7 to the MDP $\widetilde{M}_k = \langle S, [K], f, \widetilde{r}_k\rangle$ and compute the corresponding optimal (optimistic) policy $\widetilde{\pi}_k$. It is simple to verify that $(\widetilde{M}_k, \widetilde{\pi}_k)$ is the pair of MDP and policy that maximizes the average reward over all "plausible" MDPs that are within the confidence intervals over the reward function. More formally, let $\mathcal{M}_k = \{M = \langle S, [A], f, r\rangle,\ |r(s,a) - \widehat{r}_t(s,a)| \leq c_{t,a}\|x_{s,a}\|_{V_{t,a}^{-1}}, \forall s,a\}$,

Algorithm 10 The LinUCRL algorithm.

Notation: $c_{t,a} = R\sqrt{(d+1)\log\left(Kt^\alpha\left(1+\frac{t_a L_w^2}{\lambda}\right)\right)} + \lambda^{1/2}B$

Init: Set $t=0$, $T_a=0$, $\widehat{\theta}_a = \mathbf{0} \in \mathbb{R}^{d+1}$, $V_a = \lambda I$

for rounds $k = 1, 2, \cdots$ **do**

  Initialize round k

  Set $t_k = t$, $\nu_a = 0$

  Compute $\widehat{\theta}_a = V_a^{-1} X_a^\mathsf{T} R_a$

  Set optimistic reward $\widetilde{r}_k(s,a) = x_{s,a}^\mathsf{T}\widehat{\theta}_a + c_{t,a}\|x_{s,a}\|_{V_a^{-1}}$

  Compute optimal policy $\widetilde{\pi}_k$ for MDP $(S, [K], f, \widetilde{r}_t)$

  Execute chosen policy $\widetilde{\pi}_{t_k}$

  while $\forall a \in [K], T_a < \nu_a$ **do**

    Choose action $a_t = \widetilde{\pi}_k(s_t)$

    Observe reward r_t and next state s_{t+1}

    Update $X_{a_t} \leftarrow [X_{a_t}, x_{s_t,a_t}]$, $R_{a_t} \leftarrow [R_{a_t}, r_t]$, $V_{a_t} \leftarrow V_{a_t} + x_{s_t,a_t}x_{s_t,a_t}^\mathsf{T}$

    Set $\nu_{a_t} \leftarrow \nu_{a_t} + 1$, $t \leftarrow t+1$

  end while

  Set $T_a \leftarrow T_a + \nu_a, \forall a \in [K]$

end for

then we have that
$$\eta^{\widetilde{\pi}_k}(\widetilde{M}_k) \geq \max_{\pi, M \in \mathcal{M}_k} \eta^\pi(M).$$
Finally, LinUCRL execute $\widetilde{\pi}_k$ until the number of samples for an action is doubled w.r.t. the beginning of the episode. The specific structure of the problem makes LinUCRL more efficient than UCRL, since each iteration of Eq. 5.4 thanks to Eq.5.7 has $O(dSK)$ computational complexity compared to $O(S^2K)$ of extended value iteration (used in UCRL) due to the randomness of the transitions and the optimism over f.

Theoretical Analysis. We prove that LinUCRL successfully exploits the structure of the problem and reduce its cumulative regret w.r.t. basic UCRL. We first make explicit the confidence interval in Eq. 5.10. Let assume that there exist (known) constants B and R such that $\|\theta_a^*\|_2 \leq B$ for all actions $a \in [K]$ and the noise is sub-Gaussian with parameter R. Let $\ell_w = \log(w)+1$, where w is the length of the window in the state definition, and $L_w^2 = \frac{1-\ell_w^{d+1}}{1-\ell_w}$, where d is the degree of the polynomial describing the reward function. Then, we run LinUCRL with the scaling factor
$$c_{t,a} = R\sqrt{(d+1)\log\left(Kt^\alpha\left(1+\frac{T_{t,a}L_w^2}{\lambda}\right)\right)} + \lambda^{1/2}B \tag{5.11}$$
where $T_{t,a}$ is the number of samples collected from action a up to t. Then we can prove the following.

Theorem 10. *If* LinUCRL *runs with a scaling factor in Eq. 5.11 over T rounds, then its cumulative regret is bounded as*

$$\Delta(\text{LinUCRL}, T) \leq Kw \log_2\left(\frac{8T}{K}\right) + 2c_{\max}\sqrt{2KT \log\left(1 + \frac{TL_w^2}{\lambda(d+1)}\right)},$$

where $c_{\max} = \max_{t,a} c_{t,a}$ which is obtained by replacing $T_{t,a}$ by T in Eq. 5.11.

We first notice that the per-step regret Δ/T decreases to zero as $1/\sqrt{T}$, showing that as time increases, the reward approaches the optimal average reward. Furthermore, by leveraging the specific structure of our problem, LinUCRL greatly improves the dependency on other elements characterizing the MDP. In the general MDP case, UCRL suffers from a regret $O(DS\sqrt{KT})$, where D is the diameter of the MDP, which in our case is equal to the history window w. In the regret bound of LinUCRL the dependency on the number of states (which is exponential in the history window $S = K^w$) disappears and it is replaced by the number of parameters $d + 1$ in the reward model. Furthermore, since the dynamics is deterministic and known, the only dependency on the diameter w is in a lower-order logarithmic term. This result suggests that we can take a large window w and a complex polynomial expression for the reward (i.e., large d) without compromising the overall regret. Furthermore, LinUCRL compares favorably with a linear bandit approach. First, η^* is in general much larger than the optimal average reward of a greedy policy selecting the best instantaneous action at each step. Second, apart from the $\log(T)$ term, the regret is the same of a linear bandit algorithm (e.g., LinUCB). This means that LinUCRL approaches a better target performance η^* almost at the same speed as linear bandit algorithms reach a worse greedy policy.

In order to prove Thm. 10, we first need the following proposition about the confidence intervals used in computing the optimistic reward $\widetilde{r}(s, a)$.

Proposition 11. *Let assume $\|\theta_a^*\|_2 \leq B$. If $\widehat{\theta}_{t,a}$ is computed as in Eq. 5.9 and $c_{t,a}$ is defined as in Eq. 5.11, then*

$$\mathbb{P}\left(\widetilde{r}(s, a) < r(s, a) - c_{t,a}\|x_{s,a}\|_{V_{t,a}^{-1}}\right) \leq \frac{t^{-\alpha}}{K}$$

Proof. By definition of $\rho(s, a)$ we have $0 \leq \rho(s, a) \leq \sum_{\tau=1}^{w} \frac{1}{\tau} < \log(w) + 1 \doteq \ell_w$. Thus $1 \leq \|x_{s,a}\|_2^2 \leq \sum_{j=0}^{d} \ell_w^j = \frac{1-\ell_w^{d+1}}{1-\ell_w} = L_w^2$. Using Thm. 2 of [Abbasi-yadkori et al., 2011], we have with probability $1 - \delta$,

$$\|\widehat{\theta}_{t,a} - \theta_a^*\|_{V_{t,a}} \leq R\sqrt{(d+1)\log\left(\frac{1 + T_{t,a}L_w^2/\lambda}{\delta}\right)} + \lambda^{1/2}B$$

Thus for all $s \in S$ we have,

$$\begin{aligned}
|r(s,a) - \widehat{r}(s,a)| &= |x_{s,a}^{\mathsf{T}} \widehat{\theta}_{t,a} - x_{s,a}^{\mathsf{T}} \theta_a^*| \\
&= |x_{s,a}^{\mathsf{T}} V_{t,a}^{-1} V_{t,a} (\widehat{\theta}_{t,a} - \theta_a^*)| \\
&\leq \|x\|_{V_{t,a}^{-1}} \|V_{t,a}(\widehat{\theta}_{t,a} - \theta_a^*)\|_{V_{t,a}^{-1}} \quad \text{using Cauchy-Schwarz} \\
&\leq \|x\|_{V_{t,a}^{-1}} \|\widehat{\theta}_a - \theta_a^*\|_{V_{t,a}} \quad \text{since } ||V_{t_a,a}(\hat{\phi}_a - \phi_a)||_{V_{t_a,a}^{-1}} = ||\hat{\phi}_a - \phi_a||_{V_{t_a,a}}
\end{aligned}$$

Using $\delta = \frac{t^{-\alpha}}{K}$ concludes the proof.

An immediate result of Prop. 11 is that the estimated average reward of $\widetilde{\pi}_k$ in the optimistic MDP $\widetilde{M}_k$ is an upper-confidence bound on the optimal average reward, i.e., for any t (the probability follows by a union bound over actions)

$$\mathbb{P}\big(\eta^* > \eta^{\widetilde{\pi}_k}(\widetilde{M}_k)\big) \leq t^{-\alpha}. \tag{5.12}$$

We are now ready to prove the main result.

Proof of Thm. 10. We follow similar steps as in [Jaksch et al., 2010]. We split the regret over episodes as

$$\Delta(\mathcal{A}, T) = \sum_{k=1}^{m} \sum_{t=t_k}^{t_{k+1}-1} \big(\eta^* - r(s_t, a_t)\big) = \sum_{k=1}^{m} \Delta_k.$$

Let $\mathcal{T}_{k,a} = \{t_k \leq t < t_{k+1} : a_t = a\}$ be the steps when action a is selected during episode k. We upper bound the per-episode regret as

$$\begin{aligned}
\Delta_k &= \sum_{a \in [K]} \sum_{t \in \mathcal{T}_{k,a}} \big(\eta^* - r(s_t, a)\big) \\
&\leq \sum_{t=t_k}^{t_{k+1}-1} \big(\widetilde{\eta}_k - \widetilde{r}_k(s_t, a)\big) && (5.13) \\
&\quad + \sum_{a \in [K]} \sum_{t \in \mathcal{T}_{k,a}} \big(\widetilde{r}_k(s_t, a) - r(s_t, a)\big), && (5.14)
\end{aligned}$$

where the inequality directly follows from the event that $\widetilde{\eta}_k \geq \eta^*$ (Eq. 5.12) with probability $1 - T^{-\alpha}$. Notice that the low-probability event of failing confidence intervals can be treated as in [Jaksch et al., 2010].

We proceed by bounding the term in Eq. 5.13. Since we are dealing with deterministic MDPs, the optimal policy converges to a loop over states. When starting a new policy, we may start from a state outside its loop. Nonetheless, it is easy to verify that starting from any state s, it is always possible to reach any desired state s' in at most w steps (i.e., the size of the history window). As a result, the difference between the cumulative reward $(\sum_t \widetilde{r}_k(s_t, a))$ and the optimal cumulative reward $((t_{k+1} - t_k)\widetilde{\eta}_k)$ in the loop never exceeds w. Furthermore, since episodes terminate when one action doubles its number of samples, using a similar proof as [Jaksch et al., 2010], we have that the number of episodes is bounded as $m < K \log_2(\frac{8T}{K})$. As a result, the contribution of Eq. 5.13 to the overall regret can be bounded as

$$\sum_{k=1}^{m} \sum_{t=t_k}^{t_{k+1}-1} \big(\widetilde{\eta}_k - \widetilde{r}_k(s_t, a)\big) \leq K w \log_2\Big(\frac{8T}{K}\Big). \tag{5.15}$$

The second term in Eq. 5.14 refers to the (cumulative) estimation error in the reward function. We further decompose the error as

$$|\widetilde{r}_k(s_t, a) - r(s_t, a)| \leq |\widetilde{r}_k(s_t, a) - \widehat{r}_k(s_t, a)| + |\widehat{r}_k(s_t, a) - r(s_t, a)|.$$

We can bound the cumulative sum of the second term as (similar for the first, since $\widetilde{r}_k$ belongs to the confidence interval of $\widehat{r}_k$ by construction)

$$\begin{aligned}
\sum_{k=1}^{m} \sum_{a\in[K]} \sum_{t\in\mathcal{T}_{k,a}} |\widehat{r}_k(s_t, a) - r(s_t, a)| &\leq \sum_{k=1}^{m} \sum_{a\in[K]} \sum_{t\in\mathcal{T}_{k,a}} c_{t,a} \|x_{s_t,a}\|_{V_{a,t}^{-1}} \\
&\leq \max_{t,a} c_{t,a} \sum_{a\in[K]} \sum_{k=1}^{m} \sum_{t\in\mathcal{T}_{k,a}} \|x_{s_t,a}\|_{V_{a,t}^{-1}} \\
&\leq c_{\max} \sum_{a\in[K]} \sqrt{\sum_{k=1}^{m} \sum_{t\in\mathcal{T}_{k,a}} \|x_{s_t,a}\|^2_{V_{a,t}^{-1}}} \sqrt{T_a},
\end{aligned}$$

where the first inequality follows from Prop. 11 with probability $1 - T^{-\alpha}$, and T_a is the total number of times a has been selected at step T. Let $\mathcal{T}_a = \cup_k \mathcal{T}_{k,a}$, then using Lemma 11 of [Abbasi-yadkori et al., 2011], we have

$$\sum_{t\in\mathcal{T}_a} \|x_{s_t,a}\|^2_{V_{t,a}^{-1}} \leq 2 \log \frac{\det(V_{T,a})}{\det(\lambda I)}.$$

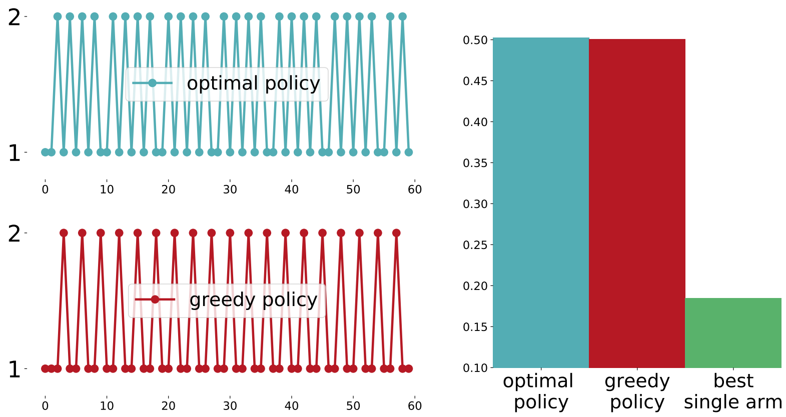
Furthermore by Lemma 10 of [Abbasi-yadkori et al., 2011], we have

$$\det(V_{t,a}) \leq (\lambda + t L_w^2/(d+1))^{d+1},$$

which leads to

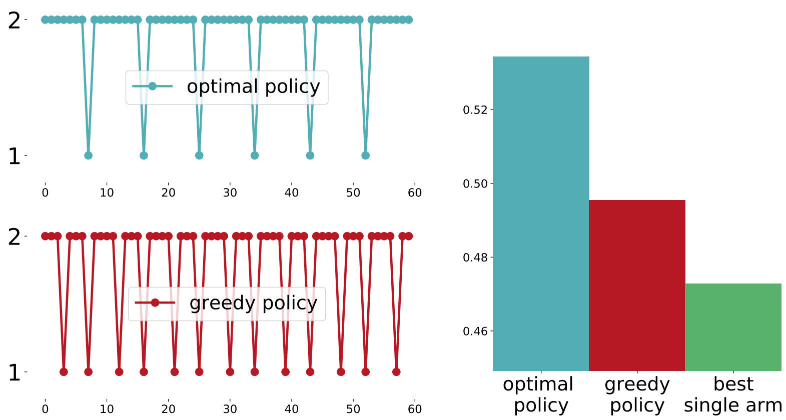
$$\begin{aligned}
\sum_{k=1}^{m} \sum_{a\in[K]} \sum_{t\in\mathcal{T}_{k,a}} |\widehat{r}_k(s_t, a) - r(s_t, a)| &\leq c_{\max} \sum_{a\in[K]} \sqrt{T_a} \sqrt{2(d+1) \log\Big(\frac{\lambda + t L_w^2}{\lambda(d+1)}\Big)} \\
&\leq c_{\max} \sqrt{2KT(d+1) \log\Big(\frac{\lambda + t L_w^2}{\lambda(d+1)}\Big)}.
\end{aligned}$$

Bringing all the terms together gives the regret bound.



(sequence of actions) (average reward)

Figure 5.4: Optimal policy vs. greedy and fixed-action. The fixed-action policy selects the action with the largest "constant" reward. The greedy policy selects the action with the highest immediate reward (depending on the state). The optimal policy is computed with value iteration. Parameters $c_1 = 0.3, c_2 = 0.4, \alpha = 1.5$.



(sequence of actions) (average reward)

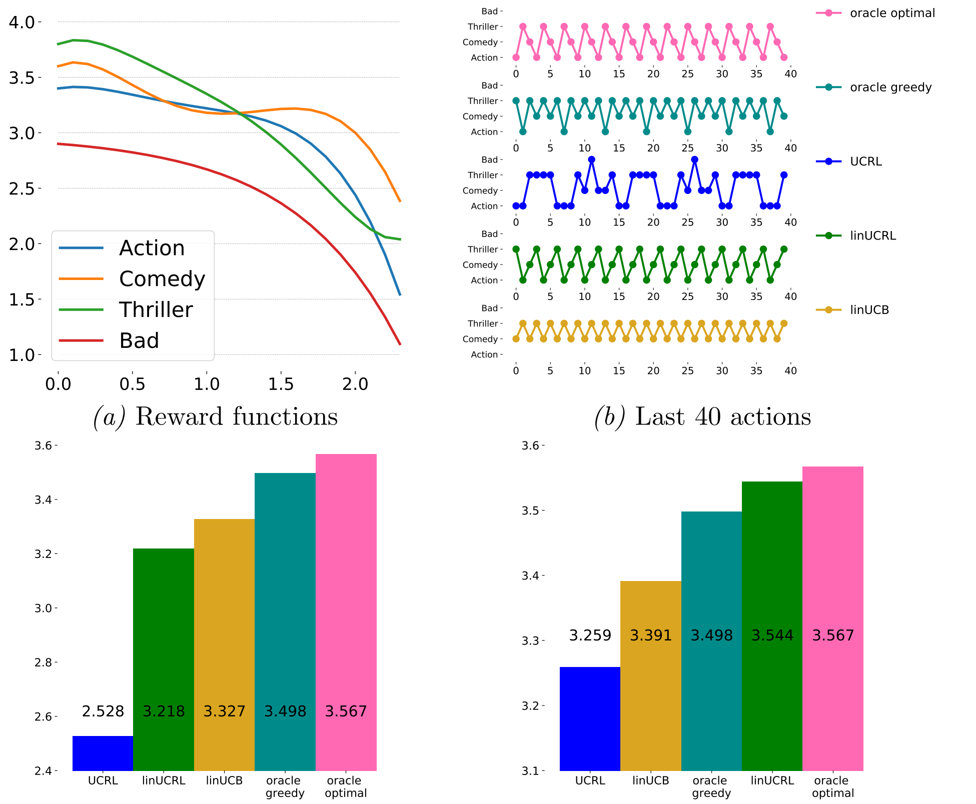
Figure 5.5: Optimal policy vs. greedy and fixed-action. The fixed-action policy selects the action with the largest "constant" reward. The greedy policy selects the action with the highest immediate reward (depending on the state). The optimal policy is computed with value iteration. Parameters $c_1 = 2, c_2 = 0.01, \alpha = 2$.

5.6 Experiments

We empirically validated our model and compared LinUCRL to existing baselines in three different scenarios. *Toy experiment:* A simulated environment with two actions and different parameters, with the objective of illustrating when the optimal policy could outperform a fixed-action and greedy strategies. *Movielens:* We derived model parameters from the *movielens* dataset and we compared the learning performance (i.e., cumulative reward) of LinUCRL to baseline algorithms. *Real-world A/B testing data:* We used a real-world A/B testing dataset where our model assumptions are no longer satisfied. We investigated how a long-term policy alternating A and B on the basis of past choices can outperform each solution individually. All codes are available at [Warlop, 2017].

Optimal vs. fixed-action and greedy policy. We first illustrate the potential improvement coming from a non-static policy that takes into consideration the recent sequence of actions and maximizes the long-term reward, compared to a greedy policy that selects the action with the higher immediate reward at each step. Intuitively, the gap may be large whenever an action has a large instantaneous reward that decreases very fast as it is selected (e.g., boredom effect). A long-term strategy may prefer to stick to selecting a sub-optimal action for a while, until the better action goes back to its initial value. We consider the simple case $K = 2$, $w = 8$ and $d = 1$. Let $\theta_1^* = (1, -c_1), \theta_2^* = (1/\alpha, -c_2)$. We study the optimal policy maximizing the average reward η, a greedy policy that always selects $a_t = \arg\max_a r(s_t, a)$, and a fixed-action policy $a_t = \arg\max\{1 - c_1\rho_M, 1/\alpha - c_2\rho_M\}$ with $\rho_M = \sum_{\tau=1}^{w} \frac{1}{\tau}$, that is the action that have the highest mean reward when always pulled. Notice that the fixed-action policy is what a recommender system would execute once the A/B testing phase is over and the (supposedly) best strategy is determined. We first set $c_1 = 0.3 \approx c_2 = 0.4$ and $\alpha = 1.5$, for which the "boredom" effect (i.e., the decrease in reward) is very mild. In this case (see Fig. 5.4), the fixed-action policy performs very poorly, while greedy and optimal policy smartly alternates between actions so as to avoid decreasing the reward of the "best" action too much. In this case, the difference between greedy and optimal policy is very narrow. In fact, switching action as soon as the instantaneous reward is better, already allows an action to "recharge" its reward enough to obtain an overall good performance. However in Fig. 5.5, with $c_1 = 2 \gg c_2 = 0.01$ and $\alpha = 2$, we see that the greedy policy switches to action 1 too soon to gain immediate reward (plays action 1 for 22% of the time) whereas the optimal policy stick to action 2 longer (plays action 1 for 11% of the time) so as to allow action 1 to regain reward and then go back to select it again. Indeed the "fresh" reward - that is the reward when the arm as not been pulled since w steps - of first arm is twice as big as the second one but it decreases really fast. It then must be pick at a very precise cadence to keep providing positive rewards. As a result, the optimal policy exploits the full potential of action 1 better and eventually gains higher average reward. While here we only leveraged on the "boredom" effect, with a reward linearly decreasing with the recency, we can imagine large range of scenarios where the greedy policy is highly suboptimal compared to the optimal policy. For instance reward function in a inverse U-shape could pose serious problem for short term strategy to successfully stay in the peak area.

Learning on *movielens* dataset. The standard offline evaluation of a learning algorithm on historical data, is to use a replay or counterfactual strat-



(a) Reward functions (b) Last 40 actions

(c) Avg. rwd. at $T = 200$ (d) Avg. rwd. at the end

Figure 5.6: Results of learning experiment based on *movielens* dataset.

egy [Li et al., 2011, Swaminathan and Joachims, 2015], which consist in updating the model whenever the learning algorithm takes the same action as in the logged data, and only update the state (but not the model) otherwise. In our case this replay strategy can not be applied because the reward depends on the history of selected actions and we could not evaluate the reward of action if the algorithm generated a sequence that is not available in the dataset, which is quite likely, and thus the model will be very rarely updated. For instance if at one step the model choose a different action than the one in the logged data, the model will only be updated in best case scenario in w steps if the next w choices are exactly the same as in the logged data. Thus in order to compare the learning performance of LinUCRL to existing baselines, we use the *movielens100k* dataset to estimate the parameters of our model and construct the corresponding "simulator". That is we first fit our model in order to predict the average observed reward for a genre for each value of the recency function based on all available data. Then whenever an arm is pulled in a specific context the simulator return the "true" reward according to equation 5.5 plus a Gaussian noise, but of course the parameter of the simulator are kept

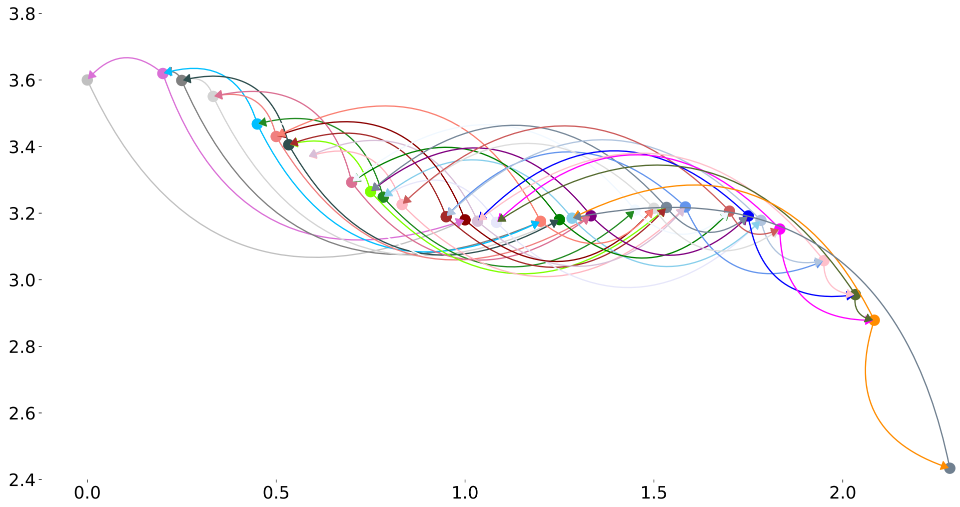
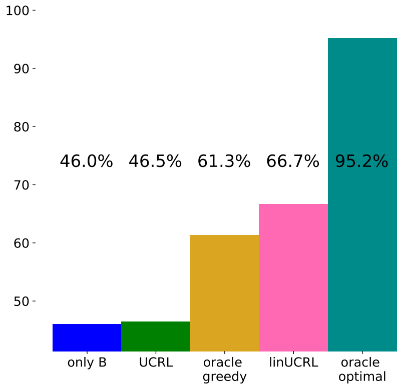


Figure 5.7: Impact of pulling an arm on next reward on movielens dataset. Here we focus on the comedy genre. Each dot represent a particular value of the recency function (equivalently a state) of comedy in the x-axis and correspond reward value if playing comedy in this state in the y-axis. Each arrow going to the right (resp. eft) indicates what will be the next state and reward of comedy if comedy is pulled (resp. not pulled) in the current state.

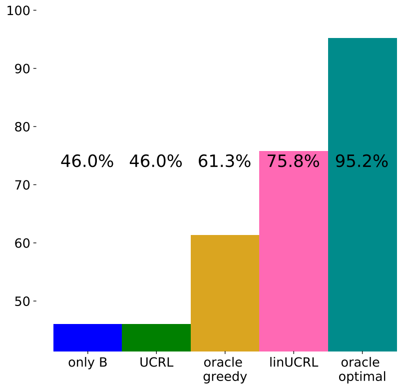
unkown to all the learning algorithm. Unlike a fully synthetic experiment, this gives a configuration which is "likely" to appear in practice, as the parameters are directly estimated from real data. We choose $K = 4$ actions corresponding to three different genres of movies plus one arbitrary bad action, and we set $d = 5$ and $w = 5$, which results into $K^w = 1024$ states. We recall that w has a mild impact on the learning performance of LinUCRL as it does not need to repeatedly try the same action in each state (as UCRL) to be able to estimate its reward. This is also confirmed by the regret analysis that shows that the regret only depends on w in the lower-order logarithmic term of the regret. The parameters that described the dependency of the reward function on the recency (i.e., $\theta^*_{j,a}$ for $j > 0$) are computed by using the ratings averaged over all users for each state encountered and for three different genres in the dataset. The first component of the vectors θ^*_a is chosen to simulate different user's preferences and to create complex dynamics in the reward functions. The resulting parameters are $\theta^*_{\text{action}} = [3.4, 0.24, -1.2, 1.2, -0.45, 0.03]$, $\theta^*_{\text{comedy}} = [3.6, 0.66, -3.48, 3.6, -1.35, 0.15]$, $\theta^*_{\text{thriller}} = [3.8, 0.6, -2.85, 3.12, -1.62, 0.3]$. The additional "bad" genre (e.g., section children for an adult user) is defined with $\theta^*_{\text{bad}} = [2.9, -0.1, -0.1, -0.01, -0.01, -0.01]$ to introduce diversity in the action set. Again, the observed reward is obtained by adding a small random Gaussian noise to the linear function. The resulting rewards as a function of the recency are illustrated in Fig. 5.6*(a)* and as a Markov Chain in Fig. 5.7 for the comedy genre. We notice that apart from the "bad" action, the reward slightly improves as a genre is seen a bit more (we can see that by noticing that $\theta^*_{a,1} > 0$ which is the predominant coefficient when the recency function is lower than 1, that is the arm has been pulled only a while ago, so a positive impact occurs) and then they all tend to decrease (with different slopes) implying that the user gets bored ($\theta^*_{a,2} < 0$ which is the predominant coefficient when the recency function is greater than 1). The results are obtained by averaging 4 independent runs.

We compare LinUCRL to the following algorithms: *oracle optimal* (optimal policy given the true parameters θ^*_a), *oracle greedy* (greedy policy given the true parameters θ^*_a), LinUCB [Abbasi-yadkori et al., 2011] (learn the parameters using LinUCB for each action and select the one with largest instantaneous reward), UCRL [Auer et al., 2009] (considering each action and state independently). Fig. 5.6*(*c-d) shows the average reward at the beginning of the learning process up to $T = 200$ and after $T = 2000$ steps. We first notice that as in the previous experiment the oracle greedy policy is suboptimal compared to the optimal policy that maximizes the long-term reward. Despite the fact that UCRL targets the optimal policy, the learning process is very slow as the number of states is too large and even after 2000 steps the learned policy (Fig. 5.6*(b)*) is still erratic, implying that the algorithm is far from convergence. On the other hand, at early learning stages LinUCRL is slightly worse than LinUCB, as it is trying to learn a more complex policy. Nonetheless, its performance rapidly improves and after 2000 steps the average reward is already better than LinUCB and it closely approaches the average reward of the optimal policy (3.567 for the optimal policy vs. 3.544 for LinUCRL). Qualitatively, we see that the greedy policy selects *thriller* and *comedy* - that have the highest immediate rewards - too often, which prevents from exploiting their full potential, that could be achieved by selecting *action* a few times more as done by LinUCRL. Similarly, LinUCB gets stuck in selecting only *thriller* and *comedy*, while LinUCRL executes almost the optimal policy by integrating *action* into the loop of genres proposed to the user.

Large scale A/B testing dataset. We also validate our approach on another real world A/B testing dataset. We collected 15 days of click on ads history of a Criteo's test, where users have been proposed two variations on the display denoted as A and B. User can either click or not click on the display, which result in a binary reward of 1 for a click and 0 otherwise. Unlike a classical A/B testing each unique user has been exposed to *both* A and B but with different frequencies. This dataset is formed of 350M tuples *(user id, timestamp, version, click)* and will be released for the community as soon as possible. Remark that the system is already heavily optimized and that even a small improvement in the click-rate is very desirable. As in the *movielens* experiment it is not relevant to use replay strategy because model updates will be very rare, but thanks to the large size of the dataset, we can define an experiment close to real conditions where the experiment should excatly follow logged data ordering. To do so, whenever an arm (A or B) is pulled by the algorithm in a given state (we set $w = 10$, for a total of $2^{10} = 1024$ states), we



(a) Avg. reward on the T steps



(b) Avg. reward at the end

Figure 5.8: Relative improvement over *only A* of learning experiment based on *large scale A/B testing* dataset.

simulate a reward according to a Bernoulli with probability corresponding to the observed click rate of this action in this state. This define a new simulator based on the data, but unlike the previous experiment where we fit a simulator according to our model, here we do not impose any linear assumption on the simulator (as in Eq. 5.5) Thus with this simulator, if the environment does not follow our model we will not be able to correctly learn our model and thus make suboptimal choices. Using this simulator we compute oracle greedy and optimal policies and we compare LinUCB, LinUCRL, which is no longer able to learn the “true” model, since it does not satisfy the linear assumption, and UCRL, which may suffer from the large number of states but targets a model with potentially better performance. Indeed UCRL does not impose any regularity on the environment thus it can fit any behaviour with a sufficient amont of time. We report the results (averaged over 5 runs) as a relative improvement to the worst fixed option (i.e., in this case A). Since different users may have different intrinsic click probability, averaging over all users may blur the impact of context and individual users click probability. To prevent from that, we clustered users based on their exposition frequency on A and based on their global click rate on A and on B. The simulator was then learned on the largest cluster in order to have sufficient observation per state. Fig. 5.8*(a)* shows the reward averaged over $T = 2,000$ steps and Fig. 5.8*(b)* shows the performance of the policy learned at the end of the experiment. We notice that despite the fact that the simulator breaks our modelling assumptions, LinUCRL is still the most competitive algorithm as it achieves the best performance among the learning algorithms and it also outperforms the greedy policy.

5.7 Related Work

While the MAB model and regret minimization [Auer, 2003] or best-arm identification algorithms [Jamieson and Talwalkar, 2016, Soare et al., 2014] have been often proposed to learn effective recommender system, they all rely on the underlying assumption that the optimal recommendation strategy is to keep selecting one single arm. Likewise our work is also related to the linear bandit model [Li et al., 2010c, Abbasi-yadkori et al., 2011, Agrawal and Goyal, 2013b] (section 2.1.3), where the reward of an arm is a linear function of a context and an unknown target vector. Despite producing context-dependent policies, this model does not consider the influence that the arms selected over time may have on the state and thus overlook the potential of long-term reward maximization. However some works proposed non stationary settings for MAB and recommender systems. Finally, their also have been some reinforcement learning algorithms developed for the recommender system task.

5.7.1 Non stationarity in multi-arm bandit

Time-varying rewards in multi-arm bandit have been studied by [Heidari et al., 2016]. In their settings each time an arm is pulled, its reward can only increase with *decreasing marginal returns*. More formally to each arm a is associated an increasing reward function $f_a(t)$ where t indicates the number of times arm a have been pulled, such that $\forall t \geq 1, f_a(t+1) - f_a(t) \leq f_a(t) - f_a(t-1)$. Their setting can correspond to someone learning a task and thus perform better and better. However, unlike our scenario, the reward never decreases back even if the arm has not been pulled for a long time nor depends on the pulls frequency. An other limit of their setting is that they assumed the existence of a known time horizon T such that the goal is to maximize the reward from time 0 to T. But once T is reached, one does not know what to do next (one cannot use $T = \infty$). Their learning algorithm leverage the decreasing marginal returns property of each arm to compute a lower and an upper bound on the cumulative reward one would get by playing only this arm from now until T. If one arm is dominated by an other arm, meaning that is upper bound is lower then the lower bound of an other arm, it is removed from the candidate set of arms. Each candidate arm is then pulled once and lower and upper bound estimates are updated until T is reached or there is only one arm left. The authors also proposed the case with decreasing rewards but again all arms must be decreasing and always decreasing. [Komiyama and Qin, 2014] also consider a model where the reward *continuously decreases* over time whether it is pulled or not (e.g., modeling novelty effects, where new products naturally loose interest over time). Let consider arm a that first appears at time t_a and let assume their exists a set of d known decreasing functions $f_1(t), \cdots, f_d(t)$, then the reward $r_{a,t}$ of arm a at time t (here t increases whether or not the arm has been pulled, it is the *true* time) is defined by $r_{a,t} = r_a + \sum_{j=1}^{d} w_{a,j} f_j(t)$ where $w_{a,j}$ are weights to be learned and r_a the arm bias to be learned. This formulation can then be rewritten as a linear bandit as described in section 2.1.3, thus their algorithm, called time-decaying UCB, introduce the adequate confidence ellipsoid on rewards to decides which arm to pull at each step. However unlike our scenario, future rewards are not influence by past choices and they consider only simple decays whereas we let the model learn the shape of the reward evolution. This model fits into the more general case of restless bandit (see e.g., [Filippi et al., 2010, Tekin and Liu, 2012, Ortner et al., 2014]), where each arm has an (partially observable) internal state that evolves as a Markov chain independently from the arms that are selected over time.

5.7.2 Non stationarity in recommender systems

Time dependency has also been studied in RS, for instance [Koren, 2009] further extend SVD++ model (see section 3.6.1) by making bias terms and user latent factors time dependant: $b_u \to b_u(t), b_i \to b_i(t), U_{uk} \to U_{uk}(t)$ where t represent a timestep (a day in Netlfix dataset). Indeed, in the Netflix dataset [Koren, 2009] noticed that the ratings tends to increase when the user rates/see the movie a long time after his released which lead to consider a time dependent item bias. Besides some product may be more suitable for specific season of the year making the item bias periodical. On the user hand, her way of rating can be drifting over time leading to consider a time depend user bias. Also the taste of the user can also be changing over time, one user may like horror movie at a given time of her life and move to drama movie later, making her user latent factor time dependent. Finally, item characteristic are expected to be fixed over time since we do not expend the item to change over time. [Koren, 2009] proposed several modelisation of the time dependence with linear effects (i.e $b_u(t) = b_u + \alpha_u \text{sign}(t - t_u)|t - t_u|^\beta$ with t_u the mean date of rating of user u, $U_{uk}(t) = U_{uk} + \alpha_u \text{sign}(t - t_u)|t - t_u|^\beta + U_{uk,t}$, $b_i(t) = b_i + b_{i,Bin(t)}$ where $Bin(t)$ is just a group of days like month, quarter, ...), spline effects ($b_u(t) = b_u + \frac{\sum_{l=1}^{k_u} e^{-\gamma|t-t_l^u|} b_{t_l}^u}{\sum_{l=1}^{k_u} e^{-\gamma|t-t_l^u|}}$ which uniformly split the user time line into k_u periods) or periodical term (adding $b_{u,period(t)}$, $b_{i,period(t)}$). Finally the author try different combination of all those terms in order to compute $\hat{r}_{ui}(t) = \mu + b_u(t) + b_i(t) + V_i^T \left(U_u(t) + |N_u|^{-\frac{1}{2}} \sum_{j \in N_u} Y_j \right)$ and learn all the parameters by SGD on the same objective function as (3.35). [He and McAuley, 2016] extend over timeSVD++ by adding a visual interaction term, that is they augmented the user-item affinity function with a temporal visual interaction term $\theta_u(t)^T \theta_i(t)$. $\theta_i(t) \in \mathbb{R}^K$, with K the latent space dimension, represent the item visual features learn from the item image with Deep Convolutional Neural Network and further mulitply by temporal weights, and $\theta_u(t) \in \mathbb{R}^K$ account for the temporal interest of user u of those visual features.

Other works focus on time as a context, for instance [Baltrunas and Amatriain, 2009] proposed to split the timeline into different seasonality - within the day, within the week, within the year - and learn a separate model per season. Altough it showed better performance than the global model, a tensor approach would be more effective as shown in section 3 with HOALS.

5.7.3 Reinforcement Learning

The closest related work to our approach has been proposed by [Auer and Ortner, 2006, Jaksch et al., 2010] where they build an algorithm called UCRL(2). Our approach is at UCRL what linear bandit is at bandit. Indeed, UCRL is a learning algorithm for undiscounted reinforcement learning where each state is independent of each user. That is the reward of an action in a given state as no link whatsoever with other states and actions and thus learning must be done for each (action, state) independently. Their algorithm thus construct confidence interval around reward and transition probabilities based on previous pulled similar to Eq. 2.3, that are used to compute an optimistic policy. The policy is then played until an (action,state) pair has been observed twice as much from the beginning as since last update (in our case it is only one action that must be observed twice as much). This algorithm allows to always find the optimal policy since it does not assume any specific behaviour of reward but learning can be really long if the number of (state,action) is large as it is the case in our environment. This can be highlight looking at their regret bound in $O(DS\sqrt{KT\log T})$ where D is the diameter of the MDP and correspond to w in our case, K the number of actions and $S = K^w$ the number of states whereas our algorithm suffer from a regret bound in $O(\sqrt{KT\log(T)})$.

[Shani et al., 2005] proposed an MDP-based Recommender System that decide what item to recommend based on the last k items that the user saw. However their work can be seen as an early heuristic variant of UCRL algorithm. As such, it has no theoretical guarantee, while LINUCRL comes with strong regret guarantee independent from the number of states. The first limit of [Shani et al., 2005] is that in recommender systems there are millions of items and thus considering each item independently results in an extremely large state space that could never be learnt. Our solution solves the issue thanks to a linear model on the reward function. Leveraging on this structure we can solve MDPs with huge number of states, as shown by the regret, which is linear in the number of actions and window size, while [Shani et al., 2005] would suffer from an exponential regret (see [Auer and Ortner, 2006, Jaksch et al., 2010]). In their approach, they have to find alternate ways to approximate the transition matrix, as clustering on states, to deal with large state space. Finally, they completely ignore the exploration-exploitation trade-off as they solve the MDP based on estimated transitions and rewards. This may lead to an overall linear regret, i.e., failing to learn the optimal policy.

Also related to our work, [Kapoor et al., 2015] propose a semi-markov model to decide what item to recommend to a user based on her latent psychological state toward this item. They assumed two possible states: sensitization, state during which she is highly engaged with the item, and boredom, state during which she is not interested in the item. Thanks to the use of a semi-markov model, the next state of the user depends on how long she has been in the current state. However, like [Shani et al., 2005], they ignore the exploration-exploitation trade-off. Indeed, in their experiment they first gather 3 months of data per user in order to learn the model parameters.

Finally, this work can also be related to linearity assumptions in reinforcement learning that usually appears in approximate dynamic programming when doing representation approximation when state space are really large. A common approximation consist in considering that the state (or state-action) value function belong to a function space $\mathcal{F}$ that is a linear function of d features $\phi_1, \cdots, \phi_d : \mathcal{X} \times A \to \mathbb{R}$ such that $\mathcal{F} \doteq \{V_\alpha(x) = \sum_{j=1}^d \alpha_j \phi_j(x), \alpha \in \mathbb{R}^d\}$. [Williams and Baird, 1993] provide approximation error guarantee on the value function in applying the Bellman Residual Minimization which approximates the optimal value function by looking for the $V \in \mathcal{F}$ that minimizes the distance $||\mathcal{T}V - V||$. [Bertsekas and Ioffe, 1996] studied the approximation error of *approximate value iteration* that modify algorithm 9 in order to constraint V_{k+1} to belong to $\mathcal{F}$, thus $V_{k+1} = \arg\inf_{V \in \mathcal{F}} ||\mathcal{T}V_k - V||$. In order to perform value function learning, we can also mention extension of TD(λ) ([Tsitsiklis and Roy, 1996]), or Least-squares temporal difference (LSTD) learning ([Boyan, 1999, Ghavamzadeh et al., 2010]). More recently, [Osband et al., 2016] proposed a randomized least-squares value iteration (RLSVI) algorithm with linearly parametrized value functions. As LinUCRL, their algorithm is splitted into episodes where parameters are sampled according to a Bayesian posterior whose confidence interval are constructed similarly to our reward function condifence interval. However, in LinUCRL we only assumed linearity on the reward function and we do not need the overall value function to be linear. One of their experiment considered the item recommendation problem, but did not considered the impact of past recommendations on future user interest. Their experiment is as follow: let consider a user who provided binary feedbacks on items (+1 for likes and −1 for dislikes, missing values are seen as 0) store in a vector $x \in \mathbb{R}^p$, with p the number of items. The probability that this user likes item i is given by $\mathbb{P}(i|x) = \frac{1}{1+\exp(-[\beta_i \sum_{j=1}^p \gamma_{i,j} x_j])}$, with $\beta_i \in \mathbb{R}$ the item bias and $\gamma_{i,j} \in \mathbb{R}$ the "correlation" between items i and j. With such a model the state space size is exponential in the number of items to recommend which justify the need to perform approximate dynamic programming. They further create a simulator that, unlike our experiment, is not constructed using real past data, but, choose arbitrarly random values for β and γ in order to simulate user interest in a given context x and prove effectiveness of their model in learning the optimal policy while minimizing the regret compared to different bandit and reinforcement learning algorithms.

5.8 Industrial learnings

The non stationnarity of the performence of a solution is commonly accepted in the industrial world. However tests do not take into account this phenomenom for multiple reasons. The first one is the absence of solid solutions to tackle the problem. The second is that many tests are often performed for attribution purpose, that is measuring the impact of one version against one other, thus population is splitted in as many disjoints groups as versions to tests. The goal of attribution is then to decide the budget to invest for different solutions. However we think that when multiple solutions are available and running, it is a good idea to have an alternating strategy using LinUCRL, but the company may not be mature enough to implement it at scale. However LinUCRL as the nice property that every computed policy end up in a loop. Thus it is only required to stock this loop and the position of the user in the loop to decide what solution to use at next. The policy can then be updated every night if updated condition are matched, that is one action has been observed twice as much as since last update of the policy. Even though LinUCRL have not been used for a client at FIFTY-FIVE for now, we are implementing the solution for an in mall navigation recommendation where the objective is to suggest a new store to visit to a user that explicitly provide the information of the stores she is willing to go to. Using only one recommender system to complete the journey could results in locking the users in always the same journey because of lack of diversity. Indeed in such application, most users always go to see the same few stores and thus using only one recommender systems would most of the time recommend the same additional store.

5.9 Conclusion

In this chapter, we showed that estimating the influence of the recommendation strategy on the reward functions and computing a policy maximizing the reward in the long term can significantly outperform fixed-action or greedy policies. We introduced a novel learning algorithm, LinUCRL, to effectively learn such optimal policy and we prove that its regret is much smaller than for standard reinforcement learning algorithms (UCRL). We validated our model and its usefulness on the *movielens* dataset and on a novel A/B testing dataset. Our results illustrate how the optimal policy effectively alternates between different options, in order to keep the interest of the users as high as possible. Furthermore, we compared LinUCRL to a series of learning baselines on simulators satisfying our linearity assumptions (*movielens*) or not (A/B testing). A venue for future work is to extend the current model to take into consideration correlations between actions. Furthermore, given its speed of convergence, it could be interesting to run a different instance of LinUCRL per user - or group of users - in order to be able to offer personalized “boredom” curves. Finally, using different models of the reward as a function of the recency (e.g., logistic regression) could be used in case of binary rewards.

Conclusion

6.1 Contributions

Recommender Systems have been widely studied and many algorithms have been proposed to improve state-of-the-art performances. However several industrial constraints were not appropriately tackled by state-of-the-art algorithms. In this thesis we aimed at proposing novel algorithms to answer those unsolved issues or improve existing solutions.

The main contributions of this thesis to the research in recommender systems can be summarized as follows:

- **a higher order matrix factorization algorithm** using tensor factorization introduced in chapter 3 that can be used at scale in order to tackle multiple types of user-item interactions and in multiple types of context. Such tensor factorization methods are very suitable for online applications where most of the time the targeted action is very scarce but many other actions are available at the same time. Besides, context is also often an important feature to take into account especially in e-commerce application where users does not have the same behaviour during the different seasons of the year, during sales period and during holidays time. Our Hoals algorithm allows to deal with both contextual bias and multiple types of information without increasing the algorithm complexity too much and proved to increase performance. This is done by adding additional slice to a tensor that originally has two dimensions (user and item) for each new type of feedback and new context. Leveraging tensor unfolding and ALS-WR allow to effectively complete the tensor missing values.

- **a basket completion algorithm** introduced in chapter 4 that builds upon DPPs to directly model the objective that is to score the relevance of adding a given item to a cart instead of the global probability of buying a set of items as it was done in previous DPP solutions for basket completion. Our solution is both effective to train and to perform recommendation thanks to the tensor low rank assumption. Each slice of the tensor is used to compute the probability to add a particular target item according to the user basket. Using a logistic function on top of the kernel allows to separate item latent factors to predict if the target is relevant to add in the basket or not. Finally, we hope that our generalization of DPP using a logistic link function could lead to other generalizations like poisson DPP for instance with a logarithmic link function.

- **a linear reinforcement learning algorithm** introduced in chapter 4 to alternate between different solutions and fight user non stationary interest in each solution. We introduced a recency function that captures the fatigue of continuously using the same action and used it to defined a polynomial reward function in each state-action pair. The parameters of those polynomials are then learned in an optimistic in face of uncertainty idea to trade off between exploration and exploitation in order to maximize long term performances. We leveraged ideas from UCRL to provide a reinforcement learning algorithm and from LinUCB to compute confidence interval around rewards. Our LinUCRL algorithm has the advantage of having a regret independent from the number of states that can grow exponentially fast with the number of arms and to have a policy that is often a short loop and thus easy to implement in practice.

6.2 Future Works

In addition to the previous enhanced contributions of this thesis, several directions can be investigated to develop and improve this work. Possible future investigations are:

- **proof convergence of** Hoals. Convergence of matrix completion algorithm has long been an open question and several works tackled this issue (see appendix A) using mostly nuclear norm minimization and alternating least square approaches. Leveraging ideas from those works could allow to give some convergence guarantee for Hoals.

- **negative sampling for** Logistic DPP. In this thesis we used random sampling to produce negative samples examples for Logistic DPP experiments. However it could be interesting to investigate more dynamic methods to generate different negative samples as the algorithm learns to help differentiate between real candidates item and bad candidates that receive a relative high score. Ideas from generative adversarial networks can be an interested direction to improve negative sampling.

- **new applications for** Logistic DPP. For now we focused on the application of basket completion however the model can be used to other classification applications. Investigating the performance of Logistic DPP against state of the art of such applications would be interesting to have.

- **generalized DPP model**. With other types of link functions, ideas of Logistic DPP can be extended to model new problems. Using logarithmic link function can produce poisson DPP in order to model for instance the number of different products that a user may purchase according to the items she clicked on during the session. We also thought about application for games but we did not have the time to further test the model on such problems. A possible direction is to have a kernel of size the number of players in the universe (instead of the items in the catalog) and the determinant will capture the winning probability of a team. In an adversarial setting, each team will have their own subkernel matrix defined by the users in each team, and the difference of the two determinants will be proportional to the first team winning probability. When position exists in the game, player i latent factors in position p, noted $V_i^p \in \mathbb{R}^K$ can be further splitted into the multiplication of two vectors such that $V_i^p = \sum_{k=1}^K \tilde{V}_{i,k} R_{p,k}$ with $R_p \in \mathbb{R}^K$ capturing position specifities and $\tilde{V}_i \in \mathbb{R}^k$ player's i performance in such position. In such application DPP may be relevant to capture global coherence of the team.

- **more personalization in LinUCRL**. In practice one would like to learn one LinUCRL per user. However it could ask too much effort and resources to be feasible. Being able to dynamically split population into subgroups with similar fatigue patern could be a good solution. Another solution could be to define a reward as a sum of linear reward for different cluster of users and both learn the parameters of each cluster and the cluster to which each user belongs.

- **new applications for LinUCRL**. We believe that LinUCRL can be used in other applications than alternating between several recommender systems. For instance it could be used for human task scheduling in order to increase productivity by changing task at the right moment that is a sufficiently long time after the task starts for the human to have time to master the task and before getting fed up. Applications in resource balancing can also be thought about in order to change exploited area before the resource is exhausted and in order to guarantee a long term equilibrium.

- **take into account all past w actions** to model the state-action reward function instead of only the corresponding action. For instance instead of defining the reward $r(s_t, a_i)$ of action a_i in state s_t as a polynomial function of the recency function $\rho(s_t, a_i)$, it could be defined as a sum of polynomial function of all recency functions $(\rho(s_t, a_j))_{j\in[K]}$.

Bibliography

[yah, 2013] (2013). Yahoo! Webscope$^{\mathrm{TM}}$ Program. On request.

[ins, 2017] (2017). The instacart online grocery shopping dataset 2017.

[Abbasi-yadkori et al., 2011] Abbasi-yadkori, Y., Pál, D., and Szepesvári, C. (2011). Improved algorithms for linear stochastic bandits. In Shawe-Taylor, J., Zemel, R. S., Bartlett, P. L., Pereira, F., and Weinberger, K. Q., editors, *Advances in Neural Information Processing Systems 24*, pages 2312–2320. Curran Associates, Inc.

[Acar et al., 2010] Acar, E., Dunlavy, D. M., Kolda, T. G., and Mørup, M. (2010). Scalable tensor factorizations with missing data. In *SDM10: Proceedings of the 2010 SIAM International Conference on Data Mining*, pages 701–712.

[Affandi et al., 2014] Affandi, R. H., Fox, E. B., Adams, R. P., and Taskar, B. (2014). Learning the parameters of determinantal point process kernels. In *Proceedings of the 31th International Conference on Machine Learning, ICML 2014, Beijing, China, 21-26 June 2014*, pages 1224–1232.

[Agrawal et al., 1993] Agrawal, R., Imieliński, T., and Swami, A. (1993). Mining association rules between sets of items in large databases. *SIGMOD Rec.*, 22(2):207–216.

[Agrawal and Goyal, 2013a] Agrawal, S. and Goyal, N. (2013a). Thompson sampling for contextual bandits with linear payoffs. In Dasgupta, S. and McAllester, D., editors, *Proceedings of the 30th International Conference on Machine Learning*, volume 28 of *Proceedings of Machine Learning Research*, pages 127–135, Atlanta, Georgia, USA. PMLR.

[Agrawal and Goyal, 2013b] Agrawal, S. and Goyal, N. (2013b). Thompson sampling for contextual bandits with linear payoffs. In *Proceedings of the 30th International Conference on International Conference on Machine Learning - Volume 28*, ICML'13, pages III–1220–III–1228. JMLR.org.

[Auer, 2003] Auer, P. (2003). Using confidence bounds for exploitation-exploration trade-offs. *J. Mach. Learn. Res.*, 3:397–422.

[Auer et al., 2009] Auer, P., Jaksch, T., and Ortner, R. (2009). Near-optimal regret bounds for reinforcement learning. In Koller, D., Schuurmans, D., Bengio, Y., and Bottou, L., editors, *Advances in Neural Information Processing Systems 21*, pages 89–96. Curran Associates, Inc.

[Auer and Ortner, 2006] Auer, P. and Ortner, R. (2006). Logarithmic online regret bounds for undiscounted reinforcement learning. In Schölkopf, B., Platt, J. C., and Hofmann, T., editors, *NIPS*, pages 49–56. MIT Press.

[Aïmeur et al., 2008] Aïmeur, E., Brassard, G., Fernandez, J. M., and Onana, F. S. M. (2008). Alambic : a privacy-preserving recommender system for electronic commerce. *Int. J. Inf. Sec.*, 7(5):307–334.

[Baer, 2016] Baer, D. (2016). The 'filter bubble' explains why trump won and you didn't see it coming.

[Bakker and Heskes, 2003] Bakker, B. and Heskes, T. (2003). Task clustering and gating for bayesian multitask learning. *J. Mach. Learn. Res.*, 4:83–99.

[Baltrunas and Amatriain, 2009] Baltrunas, L. and Amatriain, X. (2009). Towards time-dependant recommendation based on implicit feedback. In *In Workshop on context-aware recommender systems (CARS-09).*

[Ben-Shimon et al., 2015] Ben-Shimon, D., Tsikinovsky, A., Friedmann, M., Shapira, B., Rokach, L., and Hoerle, J. (2015). Recsys challenge 2015 and the yoochoose dataset. In *Proceedings of the 9th ACM Conference on Recommender Systems*, RecSys '15, pages 357–358, New York, NY, USA. ACM.

[Bennett et al., 2007] Bennett, J., Lanning, S., and Netflix, N. (2007). The netflix prize. In *In KDD Cup and Workshop in conjunction with KDD.*

[Berkovsky et al., 2007] Berkovsky, S., Eytani, Y., Kuflik, T., and Ricci, F. (2007). Enhancing privacy and preserving accuracy of a distributed collaborative filtering. In *Proceedings of the 2007 ACM Conference on Recommender Systems*, RecSys '07, pages 9–16, New York, NY, USA. ACM.

[Bertsekas and Ioffe, 1996] Bertsekas, D. P. and Ioffe, S. (1996). Temporal differences-based policy iteration and applications in neuro-dynamic programming. Technical report.

[Bertsekas and Tsitsiklis, 1996] Bertsekas, D. P. and Tsitsiklis, J. N. (1996). *Neuro-Dynamic Programming.* Athena Scientific, 1st edition.

[Bhojanapalli et al., 2014] Bhojanapalli, S., Jain, P., and Sanghavi, S. (2014). Tighter low-rank approximation via sampling the leveraged element. *CoRR*, abs/1410.3886.

[Boyan, 1999] Boyan, J. (1999). Least-squares temporal difference learning. In *In Proceedings of the Sixteenth International Conference on Machine Learning*, pages 49–56. Morgan Kaufmann.

[Breese et al., 1998] Breese, J. S., Heckerman, D., and Kadie, C. (1998). Empirical analysis of predictive algorithms for collaborative filtering. In *Proceedings of the Fourteenth Conference on Uncertainty in Artificial Intelligence*, UAI'98, pages 43–52, San Francisco, CA, USA. Morgan Kaufmann Publishers Inc.

[Brijs et al., 1999] Brijs, T., Swinnen, G., Vanhoof, K., and Wets, G. (1999). Using association rules for product assortment decisions: A case study. In *Knowledge Discovery and Data Mining*, pages 254–260.

[Brin et al., 1997] Brin, S., Motwani, R., Ullman, J. D., and Tsur, S. (1997). Dynamic itemset counting and implication rules for market basket data. In *Proceedings of the 1997 ACM SIGMOD International Conference on Management of Data*, SIGMOD '97, pages 255–264, New York, NY, USA. ACM.

[Burke, 2002] Burke, R. (2002). Hybrid recommender systems: Survey and experiments. *User Modeling and User-Adapted Interaction*, 12(4):331–370.

[Burke, 2007] Burke, R. (2007). The adaptive web. chapter Hybrid Web Recommender Systems, pages 377–408. Springer-Verlag, Berlin, Heidelberg.

[Cai et al., 2011] Cai, D., He, X., Han, J., and Huang, T. S. (2011). Graph regularized nonnegative matrix factorization for data representation. *IEEE Trans. Pattern Anal. Mach. Intell.*, 33(8):1548–1560.

[Cai et al., 2010] Cai, J.-F., Candès, E. J., and Shen, Z. (2010). A singular value thresholding algorithm for matrix completion. *SIAM J. on Optimization*, 20(4):1956–1982.

[Candès and Recht, 2008] Candès, E. J. and Recht, B. (2008). Exact matrix completion via convex optimization. *CoRR*, abs/0805.4471.

[Candès and Tao, 2009] Candès, E. J. and Tao, T. (2009). The power of convex relaxation: Near-optimal matrix completion. *CoRR*, abs/0903.1476.

[Canny, 2002] Canny, J. (2002). Collaborative filtering with privacy via factor analysis. In *Proceedings of the 25th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '02, pages 238–245, New York, NY, USA. ACM.

[Celma, 2009] Celma, O. (2009). Last.fm dataset-1k users. goo.gl/VWQKsp.

[Celma, 2010] Celma, O. (2010). *Music Recommendation and Discovery in the Long Tail.* Springer.

[Cesa-Bianchi et al., 2013] Cesa-Bianchi, N., Gentile, C., and Zappella, G. (2013). A gang of bandits. In Burges, C. J. C., Bottou, L., Welling, M., Ghahramani, Z., and Weinberger, K. Q., editors, *Advances in Neural Information Processing Systems 26*, pages 737–745. Curran Associates, Inc.

[Chakraborty et al., 2013] Chakraborty, S., Zhou, J., Balasubramanian, V., Panchanathan, S., Davidson, I., and Ye, J. (2013). Active matrix completion. In *Data Mining (ICDM), 2013 IEEE 13th International Conference on*, pages 81–90. IEEE.

[Chen et al., 2012] Chen, D., Sain, S. L., and Guo, K. (2012). Data mining for the online retail industry: A case study of rfm model-based customer segmentation using data mining. *Journal of Database Marketing & Customer Strategy Management*, 19(3):197–208.

[Chen et al., 2017] Chen, L., Zhang, G., and Zhou, H. (2017). Improving the diversity of top-n recommendation via determinantal point process. *Large Scale Recommendation Systems Workshop*, abs/1709.05135.

[Chen, 2017] Chen, M. (2017). Efficient vector representation for documents through corruption. *CoRR*, abs/1707.02377.

[Chiang et al., 2015] Chiang, K.-Y., Hsieh, C.-J., and Dhillon, I. S. (2015). Matrix completion with noisy side information. In Cortes, C., Lawrence, N. D., Lee, D. D., Sugiyama, M., and Garnett, R., editors, *Advances in Neural Information Processing Systems 28*, pages 3447–3455. Curran Associates, Inc.

[Christoffel et al., 2015] Christoffel, F., Paudel, B., Newell, C., and Bernstein, A. (2015). Blockbusters and wallflowers: Accurate, diverse, and scalable recommendations with random walks. In *Proceedings of the 9th ACM Conference on Recommender Systems*, RecSys '15, pages 163–170, New York, NY, USA. ACM.

[Coenen, 2005] Coenen, F. (2005). Tlucs kdd implementation of cba (classification based on associations).

[Ding et al., 2006] Ding, C., Li, T., Peng, W., and Park, H. (2006). Orthogonal nonnegative matrix t-factorizations for clustering. In *Proceedings of the 12th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '06, pages 126–135, New York, NY, USA. ACM.

[Dunagan et al., 2011] Dunagan, J., Spielman, D. A., and Teng, S. (2011). Smoothed analysis of condition numbers and complexity implications for linear programming. *Math. Program.*, 126(2):315–350.

[Dwork, 2008] Dwork, C. (2008). Differential privacy: A survey of results. In *Proceedings of the 5th International Conference on Theory and Applications of Models of Computation*, TAMC'08, pages 1–19, Berlin, Heidelberg. Springer-Verlag.

[Ekstrand et al., 2011] Ekstrand, M. D., Riedl, J. T., and Konstan, J. A. (2011). Collaborative filtering recommender systems. *Found. Trends Hum.-Comput. Interact.*, 4(2):81–173.

[Elahi et al., 2013] Elahi, M., Braunhofer, M., Ricci, F., and Tkalcic, M. (2013). Personality-based active learning for collaborative filtering recommender systems. In *Proceeding of the XIIIth International Conference on AI*IA 2013: Advances in Artificial Intelligence - Volume 8249*, pages 360–371, New York, NY, USA. Springer-Verlag New York, Inc.

[Erkin et al., 2011] Erkin, Z., Beye, M., Veugen, T., and Lagendijk, R. L. (2011). Efficiently computing private recommendations. In *2011 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 5864–5867. IEEE.

[Evgeniou et al., 2005] Evgeniou, T., Micchelli, C. A., and Pontil, M. (2005). Learning multiple tasks with kernel methods. *J. Mach. Learn. Res.*, 6:615–637.

[Fazel, 2002] Fazel, M. (2002). *Matrix Rank Minimization with Applications*. PhD thesis, Stanford University.

[Filipović and Jukić, 2013] Filipović, M. and Jukić, A. (2013). Tucker factorization with missing data with application to low-n-rank tensor completion. *Multidimensional Systems and Signal Processing*, pages 1–16.

[Filippi et al., 2010] Filippi, S., Cappé, O., and Garivier, A. (2010). Optimally Sensing a Single Channel Without Prior Information: The Tiling Algorithm and Regret Bounds. *IEEE Journal of Selected Topics in Signal Processing*, 5(1):68 – 76.

[Foulds and Görür, 2013] Foulds, J. A. and Görür, D. (2013). Diverse personalization with determinantal point process eigenmixtures.

[Gartrell et al., 2016] Gartrell, M., Paquet, U., and Koenigstein, N. (2016). Bayesian low-rank determinantal point processes. In *Proceedings of the 10th ACM Conference on Recommender Systems*, RecSys '16, pages 349–356, New York, NY, USA. ACM.

[Gartrell et al., 2017] Gartrell, M., Paquet, U., and Koenigstein, N. (2017). Low-rank factorization of determinantal point processes. In *Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence, February 4-9, 2017, San Francisco, California, USA.*, pages 1912–1918.

[Gautier et al., 2017] Gautier, G., Bardenet, R., and Valko, M. (2017). Zonotope hit-and-run for efficient sampling from projection dpps. In *Proceedings of the 34th International Conference on Machine Learning, ICML 2017, Sydney, NSW, Australia, 6-11 August 2017*, pages 1223–1232.

[Ghavamzadeh et al., 2010] Ghavamzadeh, M., Lazaric, A., Maillard, O. A., and Munos, R. (2010). LSTD with random projections. In *Advances in Neural Information Processing Systems*.

[Gillenwater et al., 2012] Gillenwater, J., Kulesza, A., and Taskar, B. (2012). Discovering diverse and salient threads in document collections. In *Proceedings of the 2012 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning*, EMNLP-CoNLL '12, pages 710–720, Stroudsburg, PA, USA. Association for Computational Linguistics.

[Gillenwater et al., 2014] Gillenwater, J. A., Kulesza, A., Fox, E., and Taskar, B. (2014). Expectation-maximization for learning determinantal point processes. In Ghahramani, Z., Welling, M., Cortes, C., Lawrence, N. D., and Weinberger, K. Q., editors, *Advances in Neural Information Processing Systems 27*, pages 3149–3157. Curran Associates, Inc.

[Goldberg et al., 2001] Goldberg, K., Roeder, T., Gupta, D., and Perkins, C. (2001). Eigentaste: A constant time collaborative filtering algorithm. *Information Retrieval*, 4(2):133–151.

[Gopalan et al., 2013] Gopalan, P., Hofman, J. M., and Blei, D. M. (2013). Scalable recommendation with poisson factorization. abs/1311.1704.

[Grbovic et al., 2016] Grbovic, M., Radosavljevic, V., Djuric, N., Bhamidipati, N., Savla, J., Bhagwan, V., and Sharp, D. (2016). E-commerce in your inbox: Product recommendations at scale. *CoRR*, abs/1606.07154.

[Guillou et al., 2016] Guillou, F., Gaudel, R., and Preux, P. (2016). Large-scale bandit recommender system. In *International Workshop on Machine Learning, Optimization and Big Data*, pages 204–215.

[Gunasekar et al., 2013] Gunasekar, S., Acharya, A., Gaur, N., and Ghosh, J. (2013). *Noisy Matrix Completion Using Alternating Minimization*, pages 194–209. Springer Berlin Heidelberg, Berlin, Heidelberg.

[Hahsler et al., 2005] Hahsler, M., Gruen, B., and Hornik, K. (2005). arules – A computational environment for mining association rules and frequent item sets. *Journal of Statistical Software*, 14(15):1–25.

[Han et al., 2017] Han, I., Kambadur, P., Park, K., and Shin, J. (2017). Faster greedy MAP inference for determinantal point processes. *International Conference on Machine Learning*.

[Hardt, 2013] Hardt, M. (2013). On the provable convergence of alternating minimization for matrix completion. *CoRR*, abs/1312.0925.

[Hardt, 2014] Hardt, M. (2014). Understanding alternating minimization for matrix completion. In *Proceedings of the 2014 IEEE 55th Annual Symposium on Foundations of Computer Science*, FOCS '14, pages 651–660, Washington, DC, USA. IEEE Computer Society.

[Harper and Konstan, 2015] Harper, F. M. and Konstan, J. A. (2015). The movielens datasets: History and context. *ACM Trans. Interact. Intell. Syst.*, 5(4):19:1–19:19.

[He and McAuley, 2016] He, R. and McAuley, J. (2016). Ups and downs: Modeling the visual evolution of fashion trends with one-class collaborative filtering. *CoRR*, abs/1602.01585.

[Heidari et al., 2016] Heidari, H., Kearns, M., and Roth, A. (2016). Tight policy regret bounds for improving and decaying bandits. In *Proceedings of the Twenty-Fifth International Joint Conference on Artificial Intelligence*, IJCAI'16, pages 1562–1570. AAAI Press.

[Herlocker et al., 1999a] Herlocker, J., Konstan, J., Borchers, A., and Riedl, J. (1999a). An algorithmic framework for performing collaborative filtering. In *Proceedings of the 1999 Conference on Research and Development in Information Retrieval*.

[Herlocker et al., 1999b] Herlocker, J. L., Konstan, J. A., Borchers, A., and Riedl, J. (1999b). An algorithmic framework for performing collaborative filtering. In *Proceedings of the 22Nd Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '99, pages 230–237, New York, NY, USA. ACM.

[Hern, 2017] Hern, A. (2017). How social media filter bubbles and algorithms influence the election.

[Hidasi et al., 2015] Hidasi, B., Karatzoglou, A., Baltrunas, L., and Tikk, D. (2015). Session-based recommendations with recurrent neural networks. *CoRR*, abs/1511.06939.

[Hidasi et al., 2016] Hidasi, B., Quadrana, M., Karatzoglou, A., and Tikk, D. (2016). Parallel recurrent neural network architectures for feature-rich session-based recommendations. In *Proceedings of the 10th ACM Conference on Recommender Systems*, RecSys '16, pages 241–248, New York, NY, USA. ACM.

[Hill et al., 1995] Hill, W., Stead, L., Rosenstein, M., and Furnas, G. (1995). Recommending and evaluating choices in a virtual community of use. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '95, pages 194–201, New York, NY, USA. ACM Press/Addison-Wesley Publishing Co.

[Hipp et al., 2000] Hipp, J., Güntzer, U., and Nakhaeizadeh, G. (2000). Algorithms for association rule mining - a general survey and comparison. *SIGKDD Explor. Newsl.*, 2(1):58–64.

[Hu et al., 2013] Hu, L., Cao, J., Xu, G., Cao, L., Gu, Z., and Zhu, C. (2013). Personalized recommendation via cross-domain triadic factorization. In *Proceedings of the 22Nd International Conference on World Wide Web*, WWW '13, pages 595–606, Republic and Canton of Geneva, Switzerland. International World Wide Web Conferences Steering Committee.

[Hu et al., 2008] Hu, Y., Koren, Y., and Volinsky, C. (2008). Collaborative filtering for implicit feedback datasets. In *Proceedings of the 2008 Eighth IEEE International Conference on Data Mining*, ICDM '08, pages 263–272, Washington, DC, USA. IEEE Computer Society.

[Hulikal Keshavan, 2012] Hulikal Keshavan, R. (2012). *Efficient algorithms for collaborative filtering*. Phd, Stanford University.

[Jackson, 2017] Jackson, J. (2017). Eli pariser: activist whose filter bubble warnings presaged trump and brexit.

[Jain and Netrapalli, 2014] Jain, P. and Netrapalli, P. (2014). Fast exact matrix completion with finite samples. *CoRR*, abs/1411.1087.

[Jain et al., 2013] Jain, P., Netrapalli, P., and Sanghavi, S. (2013). Low-rank matrix completion using alternating minimization. In *Proceedings of the Forty-fifth Annual ACM Symposium on Theory of Computing*, STOC '13, pages 665–674, New York, NY, USA. ACM.

[Jaksch et al., 2010] Jaksch, T., Ortner, R., and Auer, P. (2010). Near-optimal regret bounds for reinforcement learning. *J. Mach. Learn. Res.*, 11:1563–1600.

[Jamieson and Talwalkar, 2016] Jamieson, K. G. and Talwalkar, A. (2016). Non-stochastic best arm identification and hyperparameter optimization. In *AISTATS*.

[Johnson, 2014] Johnson, C. C. (2014). Logistic matrix factorization for implicit feedback data. *Advances in Neural Information Processing Systems*, 27.

[Kapoor et al., 2015] Kapoor, K., Subbian, K., Srivastava, J., and Schrater, P. (2015). Just in time recommendations: Modeling the dynamics of boredom in activity streams. In *Proceedings of the Eighth ACM International Conference on Web Search and Data Mining*, WSDM '15, pages 233–242, New York, NY, USA. ACM.

[Kaufmann et al., 2012] Kaufmann, E., Korda, N., and Munos, R. (2012). Thompson sampling: An asymptotically optimal finite-time analysis. In *Proceedings of the 23rd International Conference on Algorithmic Learning Theory*, ALT'12, pages 199–213, Berlin, Heidelberg. Springer-Verlag.

[Keshavan et al., 2010] Keshavan, R. H., Montanari, A., and Oh, S. (2010). Matrix completion from noisy entries. *J. Mach. Learn. Res.*, 11:2057–2078.

[Keshavan et al., 2009] Keshavan, R. H., Oh, S., and Montanari, A. (2009). Matrix completion from a few entries. *CoRR*, abs/0901.3150.

[Koene et al., 2015] Koene, A., Perez, E., Carter, C. J., Statache, R., Adolphs, S., O'Malley, C., Rodden, T., and McAuley, D. (2015). *Ethics of Personalized Information Filtering*, pages 123–132. Springer International Publishing, Cham.

[Kolda and Bader, 2009] Kolda, T. G. and Bader, B. W. (2009). Tensor decompositions and applications. *SIAM Review*, 51(3):455–500.

[Komiyama and Qin, 2014] Komiyama, J. and Qin, T. (2014). *Time-Decaying Bandits for Non-stationary Systems*, pages 460–466. Springer International Publishing, Cham.

[Koren, 2008] Koren, Y. (2008). Factorization meets the neighborhood: A multifaceted collaborative filtering model. In *Proceedings of the 14th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '08, pages 426–434, New York, NY, USA. ACM, ACM.

[Koren, 2009] Koren, Y. (2009). Collaborative filtering with temporal dynamics. In *Proceedings of the 15th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, volume 53 of *KDD '09*, pages 447–456, New York, NY, USA. ACM.

[Koren et al., 2009] Koren, Y., Bell, R., and Volinsky, C. (2009). Matrix factorization techniques for recommender systems. *Computer*, 42(8):30–37.

[Kosinski et al., 2013] Kosinski, M., Stillwell, D., and Graepel, T. (2013). Private traits and attributes are predictable from digital records of human behavior. *Proceedings of the National Academy of Sciences*, 110(15):5802–5805.

[Kulesza and Taskar, 2012] Kulesza, A. and Taskar, B. (2012). *Determinantal Point Processes for Machine Learning.* Now Publishers Inc., Hanover, MA, USA.

[Lam et al., 2006] Lam, S. K. T., Frankowski, D., and Riedl, J. (2006). *Do You Trust Your Recommendations? An Exploration of Security and Privacy Issues in Recommender Systems.* Springer Berlin Heidelberg, Berlin, Heidelberg.

[Lathauwer et al., 2000] Lathauwer, L. D., Moor, B. D., and Vandewalle, J. (2000). A multilinear singular value decomposition. *SIAM J. Matrix Anal. Appl.*, 21(4):1253–1278.

[Lazaric, 2008] Lazaric, A. (2008). *Knowledge Transfer in Reinforcement Learning.* PhD thesis, Politecnico di Milano.

[Lazaric and Ghavamzadeh, 2010] Lazaric, A. and Ghavamzadeh, M. (2010). Bayesian multi-task reinforcement learning. In *Proceedings of the Twenty-Seventh International Conference on Machine Learning (ICML-2010).*

[Lazaric et al., 2008] Lazaric, A., Restelli, M., and Bonarini, A. (2008). Transfer of samples in batch reinforcement learning. In McCallum, A. and Roweis, S., editors, *Proceedings of the Twenty-Fifth Annual International Conference on Machine Learning (ICML-2008)*, pages 544–551, Helsinki, Finalnd.

[Le and Mikolov, 2014] Le, Q. V. and Mikolov, T. (2014). Distributed representations of sentences and documents. *CoRR*, abs/1405.4053.

[Lee et al., 2016] Lee, J., Kim, S., Lebanon, G., Singer, Y., and Bengio, S. (2016). Llorma: Local low-rank matrix approximation. *Journal of Machine Learning Research (JMLR)*, pages 1–24.

[Lee et al., 2005] Lee, J.-S., Jun, C.-H., Lee, J., and Kim, S. (2005). Classification-based collaborative filtering using market basket data. *Expert Syst. Appl.*, 29(3):700–704.

[Li et al., 2009a] Li, B., Yang, Q., and Xue, X. (2009a). Can movies and books collaborate?: Cross-domain collaborative filtering for sparsity reduction. In *Proceedings of the 21st International Jont Conference on Artifical Intelligence*, IJCAI'09, pages 2052–2057, San Francisco, CA, USA. Morgan Kaufmann Publishers Inc.

[Li et al., 2009b] Li, B., Yang, Q., and Xue, X. (2009b). Transfer learning for collaborative filtering via a rating-matrix generative model. In *Proceedings of the 26th Annual International Conference on Machine Learning*, ICML '09, pages 617–624, New York, NY, USA. ACM.

[Li et al., 2010a] Li, L., Chu, W., and Langford, J. (2010a). An unbiased, data-driven, offline evaluation method of contextual bandit algorithms. *CoRR*, abs/1003.5956.

[Li et al., 2010b] Li, L., Chu, W., Langford, J., and Schapire, R. E. (2010b). A contextual-bandit approach to personalized news article recommendation. In *Proceedings of the 19th International Conference on World Wide Web*, WWW '10, pages 661–670, New York, NY, USA. ACM.

[Li et al., 2010c] Li, L., Chu, W., Langford, J., and Schapire, R. E. (2010c). A contextual-bandit approach to personalized news article recommendation. In Rappa, M., Jones, P., Freire, J., and Chakrabarti, S., editors, *WWW*, pages 661–670. ACM.

[Li et al., 2011] Li, L., Chu, W., Langford, J., and Wang, X. (2011). Unbiased offline evaluation of contextual-bandit-based news article recommendation algorithms. In *Proceedings of the Fourth ACM International Conference on Web Search and Data Mining*, WSDM '11, pages 297–306, New York, NY, USA. ACM.

[Li et al., 2016] Li, S., Karatzoglou, A., and Gentile, C. (2016). Collaborative filtering bandits. In *Proceedings of the 39th International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '16, pages 539–548, New York, NY, USA. ACM.

[Linden et al., 2003] Linden, G., Smith, B., and York, J. (2003). Amazon.com recommendations: Item-to-item collaborative filtering. *IEEE Internet Computing*, 7(1):76–80.

[Liu et al., 1998] Liu, B., Hsu, W., and Ma, Y. (1998). Integrating classification and association rule mining. In *Proceedings of the Fourth International Conference on Knowledge Discovery and Data Mining*, KDD'98, pages 80–86. AAAI Press.

[Mackey et al., 2011] Mackey, L. W., Jordan, M. I., and Talwalkar, A. (2011). Divide-and-conquer matrix factorization. In Shawe-Taylor, J., Zemel, R. S., Bartlett, P. L., Pereira, F., and Weinberger, K. Q., editors, *Advances in Neural Information Processing Systems 24*, pages 1134–1142. Curran Associates, Inc.

[Maksai et al., 2015] Maksai, A., Garcin, F., and Faltings, B. (2015). Predicting online performance of news recommender systems through richer evaluation metrics. In *Proceedings of the 9th ACM Conference on Recommender Systems*, RecSys '15, pages 179–186, New York, NY, USA. ACM.

[Mariet and Sra, 2015] Mariet, Z. and Sra, S. (2015). Fixed-point algorithms for determinantal point processes. *CoRR*, abs/1508.00792.

[Mary et al., 2014] Mary, J., Gaudel, R., and Preux, P. (2014). Bandits warm-up cold recommender systems. *CoRR*, abs/1407.2806.

[McSherry and Mironov, 2009a] McSherry, F. and Mironov, I. (2009a). Differentially private recommender systems: Building privacy into the net. In *Proceedings of the 15th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '09, pages 627–636, New York, NY, USA. ACM.

[McSherry and Mironov, 2009b] McSherry, F. and Mironov, I. (2009b). Differentially private recommender systems: Building privacy into the netflix prize contenders. In *Proceedings of the 15th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD)*, pages 627–636. Association for Computing Machinery, Inc.

[Mikolov et al., 2013a] Mikolov, T., Chen, K., Corrado, G., and Dean, J. (2013a). Efficient estimation of word representations in vector space. *CoRR*, abs/1301.3781.

[Mikolov et al., 2013b] Mikolov, T., Sutskever, I., Chen, K., Corrado, G. S., and Dean, J. (2013b). Distributed representations of words and phrases and their compositionality. In Burges, C. J. C., Bottou, L., Welling, M., Ghahramani, Z., and Weinberger, K. Q., editors, *Advances in Neural Information Processing Systems 26*, pages 3111–3119. Curran Associates, Inc.

[Mitov and Claassen, 2013] Mitov, V. and Claassen, M. (2013). A fused elastic net logistic regression model for multi-task binary classification. *arXiv preprint arXiv:1312.7750*.

[Moreno et al., 2012] Moreno, O., Shapira, B., Rokach, L., and Shani, G. (2012). Talmud: Transfer learning for multiple domains. In *Proceedings of the 21st ACM International Conference on Information and Knowledge Management*, CIKM '12, pages 425–434, New York, NY, USA. ACM.

[Narayanan and Shmatikov, 2006] Narayanan, A. and Shmatikov, V. (2006). How to break anonymity of the netflix prize dataset. *CoRR*, abs/cs/0610105.

[Nguyen et al., 2014] Nguyen, T. T., Hui, P.-M., Harper, F. M., Terveen, L., and Konstan, J. A. (2014). Exploring the filter bubble: The effect of using recommender systems on content diversity. In *Proceedings of the 23rd International Conference on World Wide Web*, WWW '14, pages 677–686, New York, NY, USA. ACM.

[Nickel et al., 2011] Nickel, M., Tresp, V., and Kriegel, H.-P. (2011). A three-way model for collective learning on multi-relational data. In Getoor, L. and Scheffer, T., editors, *Proceedings of the 28th International Conference on Machine Learning (ICML-11)*, ICML '11, pages 809–816, New York, NY, USA. ACM.

[Ortner et al., 2014] Ortner, R., Ryabko, D., Auer, P., and Munos, R. (2014). Regret bounds for restless markov bandits. *Theor. Comput. Sci.*, 558:62–76.

[Osband et al., 2016] Osband, I., Van Roy, B., and Wen, Z. (2016). Generalization and exploration via randomized value functions. In *Proceedings of the 33rd International Conference on International Conference on Machine Learning - Volume 48*, ICML'16, pages 2377–2386. JMLR.org.

[Pan and Yang, 2010] Pan, S. J. and Yang, Q. (2010). A survey on transfer learning. *IEEE Trans. on Knowl. and Data Eng.*, 22(10):1345–1359.

[Pan et al., 2010] Pan, W., Xiang, E. W., Liu, N. N., and 0001, Q. Y. (2010). Transfer learning in collaborative filtering for sparsity reduction. In Fox, M. and Poole, D., editors, *AAAI*. AAAI Press.

[Pariser, 2011] Pariser, E. (2011). *The Filter Bubble: What the Internet Is Hiding from You*. Penguin Group , The.

[Paterek, 2007] Paterek, A. (2007). Improving regularized singular value decomposition for collaborative filtering. In *Proc. KDD Cup Workshop at SIGKDD'07, 13th ACM Int. Conf. on Knowledge Discovery and Data Mining*, pages 39–42.

[Paudel et al., 2017] Paudel, B., Haas, T., and Bernstein, A. (2017). Fewer flops at the top: Accuracy, diversity, and regularization in two-class collaborative filtering. In *Proceedings of the Eleventh ACM Conference on Recommender Systems*, RecSys '17, pages 215–223, New York, NY, USA. ACM.

[Pazzani and Billsus, 2007] Pazzani, M. J. and Billsus, D. (2007). The adaptive web. chapter Content-based Recommendation Systems, pages 325–341. Springer-Verlag, Berlin, Heidelberg.

[Polat and Du, 2005] Polat, H. and Du, W. (2005). Svd-based collaborative filtering with privacy. In *Proceedings of the 2005 ACM Symposium on Applied Computing*, SAC '05, pages 791–795, New York, NY, USA. ACM.

[Puterman, 1994] Puterman, M. L. (1994). *Markov Decision Processes: Discrete Stochastic Dynamic Programming.* John Wiley & Sons, Inc., New York, NY, USA, 1st edition.

[Puthiya Parambath et al., 2016] Puthiya Parambath, S. A., Usunier, N., and Grandvalet, Y. (2016). A coverage-based approach to recommendation diversity on similarity graph. In *Proceedings of the 10th ACM Conference on Recommender Systems*, RecSys '16, pages 15–22, New York, NY, USA. ACM.

[Ramakrishnan et al., 2001] Ramakrishnan, N., Keller, B. J., Mirza, B. J., Grama, A. Y., and Karypis, G. (2001). Privacy risks in recommender systems. *IEEE Internet Computing*, 5(6):54–62.

[Recht, 2009] Recht, B. (2009). A simpler approach to matrix completion. *CoRR*, abs/0910.0651.

[Rendle, 2010] Rendle, S. (2010). Factorization machines. In Webb, G. I., Liu, B., Zhang, C., Gunopulos, D., and Wu, X., editors, *ICDM*, pages 995–1000. IEEE Computer Society.

[Resnick et al., 1994] Resnick, P., Iacovou, N., Suchak, M., Bergstrom, P., and Riedl, J. (1994). Grouplens: An open architecture for collaborative filtering of netnews. In *Proceedings of the 1994 ACM Conference on Computer Supported Cooperative Work*, CSCW '94, pages 175–186, New York, NY, USA. ACM.

[Ricci et al., 2010] Ricci, F., Rokach, L., Shapira, B., and Kantor, P. B. (2010). *Recommender Systems Handbook.* Springer-Verlag New York, Inc., New York, NY, USA, 1st edition.

[Romera-Paredes and Pontil, 2013] Romera-Paredes, B. and Pontil, M. (2013). A new convex relaxation for tensor completion. *CoRR*, abs/1307.4653:2967–2975.

[Russo et al., 2017] Russo, D., Roy, B. V., Kazerouni, A., and Osband, I. (2017). A tutorial on thompson sampling. *CoRR*, abs/1707.02038.

[Salakhutdinov and Mnih, 2008] Salakhutdinov, R. and Mnih, A. (2008). Bayesian probabilistic matrix factorization using markov chain monte carlo. In *Proceedings of the 25th International Conference on Machine Learning*, ICML '08, pages 880–887, New York, NY, USA. ACM.

[Schafer et al., 2007] Schafer, J. B., Frankowski, D., Herlocker, J., and Sen, S. (2007). The adaptive web. chapter Collaborative Filtering Recommender Systems, pages 291–324. Springer-Verlag, Berlin, Heidelberg.

[Sedhain et al., 2014] Sedhain, S., Sanner, S., Braziunas, D., Xie, L., and Christensen, J. (2014). Social collaborative filtering for cold-start recommendations. In *Proceedings of the 8th ACM Conference on Recommender Systems*, RecSys '14, pages 345–348, New York, NY, USA. ACM.

[Shani et al., 2005] Shani, G., Heckerman, D., and Brafman, R. I. (2005). An mdp-based recommender system. *J. Mach. Learn. Res.*, 6:1265–1295.

[Shokri et al., 2009] Shokri, R., Pedarsani, P., Theodorakopoulos, G., and Hubaux, J.-P. (2009). Preserving privacy in collaborative filtering through distributed aggregation of offline profiles. In *Proceedings of the Third ACM Conference on Recommender Systems*, RecSys '09, pages 157–164, New York, NY, USA. ACM.

[Smirnova and Vasile, 2017] Smirnova, E. and Vasile, F. (2017). Contextual sequence modeling for recommendation with recurrent neural networks. *CoRR*, abs/1706.07684.

[Soare et al., 2014] Soare, M., Lazaric, A., and Munos, R. (2014). Best-Arm Identification in Linear Bandits. In *NIPS - Advances in Neural Information Processing Systems 27*, Montreal, Canada.

[Songweiping, 2017] Songweiping (2017). Github. https://github.com/Songweiping/GRU4Rec_TensorFlow.

[Steck, 2013] Steck, H. (2013). Evaluation of recommendations: Rating-prediction and ranking. In *Proceedings of the 7th ACM Conference on Recommender Systems*, RecSys '13, pages 213–220, New York, NY, USA. ACM.

[Su and Khoshgoftaar, 2009] Su, X. and Khoshgoftaar, T. M. (2009). A survey of collaborative filtering techniques. *Adv. in Artif. Intell.*, 2009:4:2–4:2.

[Swaminathan and Joachims, 2015] Swaminathan, A. and Joachims, T. (2015). Batch learning from logged bandit feedback through counterfactual risk minimization. *J. Mach. Learn. Res.*, 16(1):1731–1755.

[Sweeney, 2002] Sweeney, L. (2002). K-anonymity: A model for protecting privacy. *Int. J. Uncertain. Fuzziness Knowl.-Based Syst.*, 10(5):557–570.

[Symeonidis et al., 2008] Symeonidis, P., Nanopoulos, A., and Manolopoulos, Y. (2008). Tag recommendations based on tensor dimensionality reduction. In *Proceedings of the 2008 ACM Conference on Recommender Systems*, RecSys '08, pages 43–50, New York, NY, USA. ACM.

[Tan et al., 2005] Tan, P.-N., Steinbach, M., and Kumar, V. (2005). Association analysis: Basic concepts and algorithms. pages 327–414.

[Tekin and Liu, 2012] Tekin, C. and Liu, M. (2012). Online Learning of Rested and Restless Bandits. *IEEE Transactions on Information Theory 58(8).*

[Teo et al., 2016] Teo, C. H., Nassif, H., Hill, D., Srinivasan, S., Goodman, M., Mohan, V., and Vishwanathan, S. (2016). Adaptive, personalized diversity for visual discovery. In *Proceedings of the 10th ACM Conference on Recommender Systems*, RecSys '16, pages 35–38, New York, NY, USA. ACM.

[Toh and Yun, 2010] Toh, K.-C. and Yun, S. (2010). An accelerated proximal gradient algorithm for nuclear norm regularized linear least squares problems. *Pacific Journal of Optimization.*

[Tomasi and Bro, 2005] Tomasi, G. and Bro, R. (2005). Parafac and missing values. *Chemometrics and Intelligent Laboratory Systems*, 75(2):163–180.

[Tsitsiklis and Roy, 1996] Tsitsiklis, J. N. and Roy, B. V. (1996). Feature-based methods for large scale dynamic programming. *Machine Learning*, 22(1-3):59–94.

[Twardowski, 2016] Twardowski, B. (2016). Modelling contextual information in session-aware recommender systems with neural networks. In *Proceedings of the 10th ACM Conference on Recommender Systems*, RecSys '16, pages 273–276, New York, NY, USA. ACM.

[Vargas and Castells, 2014] Vargas, S. and Castells, P. (2014). Improving sales diversity by recommending users to items. In *Proceedings of the 8th ACM Conference on Recommender Systems*, RecSys '14, pages 145–152, New York, NY, USA. ACM.

[Vasile et al., 2016] Vasile, F., Smirnova, E., and Conneau, A. (2016). Meta-prod2vec - product embeddings using side-information for recommendation. *CoRR*, abs/1607.07326.

[Vershik and Yakubovichn, 2001] Vershik, A. M. and Yakubovichn, Y. (2001). *Asymptotic Combinatorics with Applications to Mathematical Physics.* Springer-Verlag Berlin Heidelberg.

[Wang and Singh, 2015] Wang, Y. and Singh, A. (2015). Provably correct active sampling algorithms for matrix column subset selection with missing data. *CoRR*, abs/1505.04343.

[Warlop, 2016] Warlop, R. (2016). Github. https://github.com/RomainWarlop/RecommenderSystem.

[Warlop, 2017] Warlop, R. (2017). Github. https://github.com/RomainWarlop/ReinforcementLearning.

[Warlop, 2018] Warlop, R. (2018). Github. https://github.com/RomainWarlop/logisticDPP.

[Williams and Baird, 1993] Williams, R. and Baird, L. C. (1993). Tight performance bounds on greedy policies based on imperfect value functions. Technical report.

[Wilson et al., 2007] Wilson, A., Fern, A., Ray, S., and Tadepalli, P. (2007). Multi-task reinforcement learning: A hierarchical bayesian approach. In *Proceedings of the 24th International Conference on Machine Learning*, ICML '07, pages 1015–1022, New York, NY, USA. ACM.

[Xu et al., 2013] Xu, M., Jin, R., and Zhou, Z.-H. (2013). Speedup matrix completion with side information: Application to multi-label learning. In Burges, C. J. C., Bottou, L., Welling, M., Ghahramani, Z., and Weinberger, K. Q., editors, *Advances in Neural Information Processing Systems 26*, pages 2301–2309. Curran Associates, Inc.

[Zhou et al., 2008] Zhou, Y., Wilkinson, D., Schreiber, R., and Pan, R. (2008). Large-scale parallel collaborative filtering for the netflix prize. In *Proceedings of the 4th International Conference on Algorithmic Aspects in Information and Management*, AAIM '08, pages 337–348, Berlin, Heidelberg. Springer-Verlag.

[Zong et al., 2016] Zong, S., Ni, H., Sung, K., Ke, N. R., Wen, Z., and Kveton, B. (2016). Cascading bandits for large-scale recommendation problems. *CoRR*, abs/1603.05359.

Appendix

On Convergence of Matrix Completion

In previous chapters we show that the problem of collaborative filtering was generally studied as matrix completion problem using a small fraction of its entries. We stated that such a general problem is ill posed because it is impossible to recover missing entries without any assumption about the matrix because the degree of liberty would be $n \times p$ and thus one would need to see the entire matrix to recover it... To solve this problem we introduced the common assumption that the matrix to recover is low rank (or approximately low rank). This assumption makes sense since it implies correlation between users (and items) and that only few factors contributes to define user-item affinity. However, we never discuss if given this assumption the recovery problem was now possible nor condition on the number of entries one would need to recover a good approximation of the full matrix. In this chapter we then would like to present main works that addressed these questions in order to give the reader a better understanding of the matrix completion problem.

A.1 Background

Let consider a rank-K matrix $M \in \mathbb{R}^{n \times p}$ to be recovered from m entries, stored in the set $\mathcal{D} \doteq \{(i,j), M_{ij} \text{ is observed}\}$. A first lower bound on the minimal number of entries m needed to recover M can easily be obtained. Considering the SVD decomposition of M such that $M = USV^T$, with $U \in \mathbb{R}^{n \times K}$ the matrix with the left singular vectors, $V \in \mathbb{R}^{p \times K}$ the matrix with the right singular vectors, and $S \in \mathbb{R}^{K \times K}$ the diagonal matrix whose diagonal contains the singular values $\sigma_1 \geq \cdots \geq \sigma_K$. Then the singular values depends on K degrees of freedom, U on $(n-1)+(n-2)+\cdots+(n-K) = nK - K(K+1)/2$ degrees of freedom and likewise for V which result in $(n+p)K - K^2$ degrees of freedom. Thus,
lemma If $m < (n+p)K - K^2$, there exist an infinite number of rank-K matrices

with exactly the same entries as the one available in M. So exact recovering is impossible if $m < (n+p)K - K^2$. Actually because of the coupon collector's effect that state that to observe all the x distinct values of an urn when sampling with replacement, one must pull in average $x \log x$ times. Thus in our case m must rather be of the order of $(n+p)K \log(n+p) = O(nK \log n)$ (assuming that $n \geq p$).

In order to prove efficient matrix completion some additional hypothesis are needed. For instance, [Candès and Recht, 2008], in the extreme case of the rank-1 matrix $M = e_1 e_n^T$ (where $(e_i)_{i \in [n]}$ is the canonical basis), then

$$M = e_1 e_n^T = \begin{pmatrix} 0 & 0 & \cdots & 0 & 1 \\ 0 & 0 & \cdots & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & \cdots & 0 & 0 \end{pmatrix}$$

this matrix cannot be recovered unless almost all entries have been sampled and especially the last entry of the first row which have very low probability to be sampled if sampling is done at random. Another interesting example is proposed by [Candès and Recht, 2008]. Let consider the rank-2 matrix, $M = \sum_{k=1}^{2} \sigma_k u_k u_k^T$ with arbitrary σ_k's, $u_1 = (e_1 + e_2)/\sqrt{2}$ and $u_2 = (e_1 - e_2)/\sqrt{2}$. Then, we have

$$M = \begin{pmatrix} (\sigma_1+\sigma_2)/2 & (\sigma_1-\sigma_2)/2 & \cdots & 0 & 1 \\ (\sigma_1-\sigma_2)/2 & (\sigma_1+\sigma_2)/2 & \cdots & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & \cdots & 0 & 0 \end{pmatrix}$$

and one would also need to see almost all entries in order to recover M. These examples illustrate the fact that in order to be efficiently recovered, the singular vectors need to be sufficiently spread, that is uncorrelated or incoherent with the canonical basis. Let us then introduce commons additional assumptions for matrix completion.

Definition 13. ***(Condition number)*** *Consider a matrix M of rank-K, with singular values, $\sigma_1 \geq \cdots \geq \sigma_K > 0$. The condition number of the matrix M, denoted by κ_M is defined as $\kappa_M = \frac{\sigma_1}{\sigma_K}$*

Definition 14. ***(Incoherence of a matrix - version 1)****. A matrix $M \in \mathbb{R}^{n \times p}$ is incoherent with parameter μ if $\forall i \in [n], ||U_i||_2 \leq \mu\sqrt{\frac{K}{n}}$ and $\forall j \in [p], ||V_j||_2 \leq \mu\sqrt{\frac{K}{p}}$, where $M = USV^T$ is the SVD of M.*

Definition 15. ***(Incoherence of a matrix - version 2).*** *Let $M = USV^T$ be the SVD of M and P_U be the orthogonal projection on U. Then M is incoherent with parameter μ if $\frac{n}{K}\max_{i\in[n]}||P_U e_i||_2^2 \leq \mu$ and $\frac{p}{K}\max_{j\in[p]}||P_V e_j||_2^2 \leq \mu$ with (e_i) the standard basis .*

Remark Let M be μ incoherent according to definition 15. Then since $P_u = UU^T$, and $\frac{n}{K}\max_{i\in[n]}||P_U e_i||_2^2 = \frac{n}{K}\max_{i\in[n]}||U_i||_2^2$, we have $||U_i||_2 \leq \sqrt{\mu\frac{K}{n}}$. Thus M is $\sqrt{\mu}$ incoherent according to defintion 14.

Definition 16. ***(Strong incoherence of a matrix).*** *Let $M = USV^T$ be the SVD of M, P_U be the orthogonal projection on U and $E = UV^T$. Then M is strongly incoherent with parameter μ if*

$$\forall (i,i') \in [n]\times[n], \quad |< e_i, P_U e_{i'} > - \frac{K}{n}\mathbb{1}_{i=i'}| \leq \mu\frac{\sqrt{K}}{n}$$

$$\forall (j,j') \in [p]\times[p], \quad |< e_j, P_V e_{j'} > - \frac{K}{p}\mathbb{1}_{j=j'}| \leq \mu\frac{\sqrt{K}}{p}$$

$$\forall (i,j) \in [n]\times[p], \quad |E_{ij}| \leq \mu\sqrt{\frac{K}{np}}$$

Remark Strong incoherence implies incoherence. If M is μ-strongly incoherent, then we have $||P_U e_i||_2^2 \leq \mu\frac{\sqrt{K}}{n} + K/n = (1+\frac{\mu}{\sqrt{K}})\frac{K}{n}$. Thus μ-strongly incoherence implies $1+\frac{\mu}{\sqrt{K}}$-incoherence.

In order to give recovery condition of low-rank matrices, two researches direction emerged: nuclear norm minimization and alternating least square. The first solution as the advantage to theoretically require less samples that the second but is computationally much more greedy. Eventually, other approaches tries to reduce the number of required samples while keeping an efficient algorithm.

A.2 Nuclear norm minimization

Initially the matrix completion problem in collaborative filtering is to find the minimal rank matrix that have the same values as the initial matrix on the given points $\mathcal{D}$. More formally, let M be the sparse input matrix whose revealed entries are stocked in the set $\mathcal{D} = \{(i,j), M_{ij} \text{ is known}\}$. The problem is

$$\begin{aligned} &\text{minimize} && \text{rank}(X) \\ &\text{such that} && \forall (i,j) \in \mathcal{D}, X_{ij} = M_{ij} \end{aligned} \tag{A.1}$$

However this problem is in general an NP-hard problem. To solve this issue, several authors ([Candès and Recht, 2008, Candès and Tao, 2009, Recht, 2009]) studied the problem of nuclear norm minimization relaxation:

$$\begin{array}{ll} \text{minimize} & ||X||_* \\ \text{such that} & \forall (i,j) \in \mathcal{D}, X_{ij} = M_{ij} \end{array} \tag{A.2}$$

where $||X||_* \doteq \sum_i \sigma_i(X)$ that is the sum of the singular values. Indeed the nuclear norm can be thought as a convex relaxation of the number of non-zero eigenvalues that is the rank. This problem has the advantage of being convex and is then studied as a surrogate of the rank minimization problem.

These authors assumed different hypothesis on the input matrix M (incoherence or strong incoherence mostly) resulting in different guarantees on the reconstruction but always assumed that entries were sampled at random. Here is a short summary of their works and results.

Exact Matrix Completion via Convex Optimization

In 2008, [Candès and Recht, 2008] are the first to provide theoretical guarantee for low-rank matrix completion given partially observed matrix. The matrix is assumed to be exactly low-rank and entries are observed without noise. Under incoherence assumption, the authors gives condition on the number of entries to recover the full matrix that closely compare with the minimal theoretical value $nK \log n$ by proving that if the number of samples m is $O(n^{1.2} K \log n)$ then one can recover the input matrix with high probability.

Theorem 12 (Theorem 1.3 of [Candès and Recht, 2008]). *Let $M \in \mathbb{R}^{n\times p}$ be a rank-K matrix with singular value decomposition USV^T, and let $\bar{n} = \max(n,p)$. Assume that M is μ_0-incoherent (definition 14), and that there is $\mu_1 > 0$ such that $||UV^T||_\infty \leq \mu_1 \sqrt{r/(np)}$. Suppose entries of M are observed with locations sampled uniformly at random. If*

$$m \geq C \max(\mu_1^2, \mu_0^{1/4}\mu_1, \mu_0^{1/2} n^{1/4}) \bar{n} K \beta \log \bar{n}$$

for some $\beta > 2$ and constant $C > 0$, then the minimizer of the problem

$$\begin{array}{ll} \textit{minimize} & ||X||_* \\ \textit{such that} & \forall (i,j) \in \mathcal{D}, X_{ij} = M_{ij} \end{array}$$

is unique and equal to M with probability at least $1 - cn^{-\beta}$. For $r \leq \mu_0^{-1/2} n^{1.5}$, this estimate can be improved to $C\mu^{1/2} n^{1.2} K \beta \log \bar{n}$ with the same probability of success.

The Power of Convex Relaxation: Near-Optimal Matrix Completion

Trying to get closer to $nK \log n$ bound, in 2009 [Candès and Tao, 2009] relied on strong incoherence assumption of the matrix in order to recover the input matrix with high probability. This allowed them to further reduce the number of required samples to $O(nK \log^6 n)$ which improve the dependence on n over previous result.

Theorem 13 (Theorem 1.2 of [Candès and Tao, 2009]). *Let $M \in \mathbb{R}^{n\times p}$ be a fixed matrix of rank $K = O(1)$ such that M is μ-strongly incoherent and $\bar{n} = \max(n, p)$. Suppose we observe m entries of M with locations sample uniformly at random. Then there is a positive numerical constant C such that if*

$$m \geq C\mu^2 \bar{n} K \log^6 \bar{n}$$

then with probability at least $1 - \bar{n}^{-3}$, M is the unique solution of (A.2).

A Simpler Approach to Matrix Completion

Finally in 2009, [Recht, 2009] relaxed the strong incoherence assumption of [Candès and Tao, 2009] to a incoherence assumption on the matrix to recover and come closer to the theoretical lower bound on the number of entries up to a $\log n$ factor ($m = O(nK \log^2 n)$). Thus under the same hypothesis it is not possible to further improve this theoretical guarantees. However they assumed an additional assumptions that assumed that UV^T is bounded.

Theorem 14 (Theorem 1.1 of [Recht, 2009]). *Let $M \in \mathbb{R}^{n\times p}$ be a rank-K matrix with singular value decomposition USV^T. Without loss of generality, impose convention $n \geq p$, $S \in \mathbb{R}^{K\times K}, U \in \mathbb{R}^{n\times K}$, and $V \in \mathbb{R}^{p\times K}$. Assume that M is μ_0-incoherent (definition 14), and that there is $\mu_1 > 0$ such that $||UV^T||_\infty \leq \mu_1\sqrt{r/(np)}$. Suppose entries of M are observed with locations sampled uniformly at random. If*

$$m \geq 32\beta \max(\mu_1^2, \sqrt{\mu_0})K(n + p)\log^2(2n)$$

for some $\beta > 1$, then the minimizer of the problem

$$\begin{array}{ll} \textit{minimize} & ||X||_* \\ \textit{such that} & \forall (i, j) \in \mathcal{D}, X_{ij} = M_{ij} \end{array}$$

is unique and equal to M with probability at least $1 - 6(n + p)^{2-2\beta}\log(p) - p^{2-2\beta^{1/2}}$.

A.3 Alternating Least Square

As seen previously, in collaborative filtering application, the alternating least square (ALS) algorithm is used very oftenly for its simplicity. Indeed in order to update the latent factors of one row of U or V, one just need to perform a linear regression of dimension K. [Jain et al., 2013] are the first to provide recovery guarantee of this algorithm. In the ALS algorithm the goal is to find the two matrices $U \in \mathbb{R}^{n\times K}$ and $V \in \mathbb{R}^{p\times K}$ such that the input matrix $R = UV^T$ given a set of observed point $\mathcal{D}$. This is done by minimizing the squared loss between the observed value and the scalar product of the corresponding lines of U and V:

Algorithm 11 ALSM

Input: dataset $\mathcal{D}$, q probability to observe each entry.
Create $2\tau + 1$ subsets from $\mathcal{D}$:$\mathcal{D}_0, \mathcal{D}_1, \cdots, \mathcal{D}_{2\tau}$ each of size $|\mathcal{D}|$ with the element of $\mathcal{D}$ belonging to one of the $\mathcal{D}_t$'s with equal probability and sampled independently

Set $\hat{U}^{(0)} = SVD(\tilde{M}_{\mathcal{D}}/q, K)$ that is the top-K left singular values of $\tilde{M}_{\mathcal{D}}/q$ where unknown entries are fill with 0.
Clipping step: Set all the elements of $\hat{U}^0$ that have magnitude greater than $\frac{2\mu\sqrt{k}}{n}$ to zero and orthonormalize the columns $\hat{U}^0$ (using QR decomposition).
for $t = 0, \cdots, \tau - 1$ **do**
  $\hat{V}^{(t+1)} \leftarrow \arg\min_{V \in \mathbb{R}^{p \times k}} \sum_{(i,j) \in \mathcal{D}_{t+1}} (R_{ij} - \hat{U}_i^{(t)} V_j^T)^2$
  $\hat{U}^{(t+1)} \leftarrow \arg\min_{U \in \mathbb{R}^{n \times k}} \sum_{(i,j) \in \mathcal{D}_{\tau+t+1}} (R_{ij} - \hat{V}_j^{(t+1)} U_i^T)^2$
end for
Output: $\hat{R} = U^{\hat{(\tau)}} (V^{\hat{(\tau)}})^T$

$$\min_{U \in \mathbb{R}^{n \times K}, V \in \mathbb{R}^{p \times K}} \sum_{(i,j) \in \mathcal{D}} (R_{ij} - U_i V_j^T)^2 \tag{A.3}$$

Low-rank Matrix Completion using Alternating Minimization
In 2012, [Jain et al., 2013] were the first to provide theoretical analysis of the alternating least square algorithm for matrix completion and matrix sensing (recovering matrix M given linear measurements $b_i = \mathrm{tr}(A_i^T M)$, where $(A_i)_{1 \leq i \leq d}$ are called sensing matrices). To perform their analysis, they showed that the alternating least square can be viewed as a noisy power method to compute the singular vectors.

For analysis constraint, they proposed a modification of ALS where each update is done using a sample subset of $\mathcal{D}$ with the same cardinality (see algorithm 11) in order to be able to analyse each iteration independently, but this is not required in practice. Using this algorithm they were able to prove the following theorem:

Theorem 15 (Theorem 2.5 of [Jain et al., 2013]). *Let $M = USV^T \in \mathbb{R}^{n \times p}$ be a rank-K, μ-incoherent matrix (definition 14). Also, let each entry of M be observed uniformly and independently with probability q, if*

$$q > C \frac{\kappa_M^2 \mu^2 K^{2.5} \log n \log \frac{K||M||_F}{\varepsilon}}{p \delta_{2K}^2}$$

where $\kappa_M = \frac{\sigma_1}{\sigma_K}$ is the condition number of M, $\delta_{2k} \leq \frac{\sigma_K}{12\sigma_1}$ and $C > 0$ is a global constant. Then with high probability with $\tau \geq C' \log \frac{||M||_F}{\varepsilon}$ the outputs $\hat{U}^{(\tau)}$ and $\hat{V}^{(\tau)}$ of Algorithm 11 with input $\mathcal{D}$ satisfy

$$||M - \hat{U}^{(\tau)} (\hat{V}^{(\tau)})^T||_F \leq \varepsilon$$

Remark The algorithm require $|\mathcal{D}| \geq C\frac{\kappa_M^2\mu^2K^{2.5}n\log n\log\frac{K||M||_F}{\varepsilon}}{\delta_{2K}^2}$, that is $|\mathcal{D}| = O(\kappa_M^4K^{2.5}n\log n\log\frac{K||M||_F}{\varepsilon})$ to recover the true matrix with an error of ε.

Noisy Matrix Completion Using Alternating Minimization
Building upon the work of [Jain et al., 2013], [Gunasekar et al., 2013] are the first to provide recovery guarantee of the alternating least square algorithm under noisy observations.
Noise. Let M be the true low rank matrix and N the noise matrix such that the observed entries come from the matrix $\tilde{M} \doteq M + N$. The noise is assumed to be distributed arbitrarily but bounded by some constant $N_{\max}$: $|N_{ij}| \leq N_{\max}$.

Using this noise definition and pursuing the power method idea, [Gunasekar et al., 2013] obtained the following results regarding the recoverability of the matrix.

Theorem 16 (Theorem 2 of [Gunasekar et al., 2013]). *Let $M = USV^T \in \mathbb{R}^{n\times p}$ be a rank-K, μ-incoherent matrix (definition 14). Further, it is assumed that $N_{\max} \leq C_3\frac{\sigma_K}{n\sqrt{K}}$ and $\frac{||N^{\mathcal{D}}||_2}{q} \leq C_2\frac{\sigma_K}{\kappa_M K}$. Additionally, let each entry of $\tilde{M} = M + N$ be observed uniformly and independently with probability q, if*

$$q > C\frac{\kappa_M^4\mu^4K^7\log n\log\frac{||M||_F}{\varepsilon}}{p\delta_{2K}^2}$$

where $\kappa_M = \frac{\sigma_1}{\sigma_K}$ is the condition number of M, $\delta_{2k} \leq \frac{\sigma_K}{64\sigma_1}$ and $C > 0$ is a global constant. Then with high probability with $\tau \geq C'\log\frac{||M||_F}{\varepsilon}$ the outputs $\hat{U}^{(\tau)}$ and $\hat{V}^{(\tau)}$ of Algorithm 11 with input $\mathcal{D}$ satisfy

$$\frac{1}{\sqrt{np}}||M - \hat{U}^{(\tau)}(\hat{V}^{(\tau)})^T||_F \leq \varepsilon + 20\mu\kappa_M^2k^{1.5}\left(\frac{||N^{\mathcal{D}}||_2}{|\mathcal{D}|}\right) \leq \varepsilon + 40\mu\kappa_M^2K^{1.5}N_{\max}$$

Remark The algorithm require $|\mathcal{D}| \geq C\frac{\kappa_M^4\mu^4K^7n\log n\log\frac{||M||_F}{\varepsilon}}{\delta_{2K}^2}$, that is $|\mathcal{D}| = O(\kappa_M^6K^7n\log n\log\frac{||M||_F}{\varepsilon})$ entries to recover a noisy low-rank μ-incoherent matrix with error ε plus noise. Let note that one must have $n << K^7$ however the number of required entries is larger than the total number of entries. Unfortunately, this is rarely the case in practice where $K \approx 100$.

Understanding Alternating Minimization for Matrix Completion
In 2014, [Hardt, 2013] reduced the number of needed entries in order to recover a low-rank matrix using an alternating minimization algorithm. The improvement is achieved using a careful analysis of QR factorization (which is used in [Jain et al., 2013]) and proposing a smooth version inspired by

[Dunagan et al., 2011]. They discussed the fact that applying QR-factorization directly to the singular vector matrix can lead to large incoherent parameter of the matrix (so to a more coherent matrix) and thus increase the number of samples needed to recover the matrix. However small Gaussian perturbations have proved to lead to well-conditioned matrix, which was no harm for their analysis since the matrix to recover is already noisy. Their alternating algorithm uses this trick to prove a low coherence number at each iteration of the alternating scheme.
Noise. Let $\tilde{M} = M + N$ where M is the low-rank matrix to recover such that $M = USU^T$, and N the noise matrix. Thus $N = (I - UU^T)\tilde{M}$. N is suppose to be an arbitrary deterministic matrix that satisfies two constraints:

$$\max_{i\in[n]} ||e_i^T N||^2 \leq \frac{\mu_N}{n}\sigma_k^2 \quad \text{and} \quad \max_{j\in[n]} |N_{ij}| \leq \frac{\mu_N}{n}||e_i^T N|| \tag{A.4}$$

This assumption is weaker than the noise assumption of [Gunasekar et al., 2013] since if $\max_{ij}|N_{ij}| \leq N_{\max}$ then constraints (A.4) are also verified.

Theorem 17 (Theorem 1.2 of [Hardt, 2014]). *Let $M = USU^T \in \mathbb{R}^{n\times n}$ be a rank-K, μ-incoherent matrix (definition 14) and $\tilde{M} = M + N$ with the noise $N = (I - UU^T)\tilde{M}$ satisfying (A.4) and whose largest singular values is denoted σ_{k+1}. Additionally, let each entry of $\tilde{M}$ be observed uniformly and independently with probability q, if*

$$q > \frac{k}{n}(k + \log(n/\varepsilon)\mu\left(\frac{||M||_F + ||N||_F/\varepsilon}{\sigma_k}\right)^2\left(1 - \frac{\sigma_{k+1}}{\sigma_k}\right)$$

then the alternating least square algorithm output (X, Y) such that $||(I - UU^T)X|| \leq \varepsilon$ and $||M - XY^T||_F \leq \varepsilon||\tilde{M}||_F$,

Remark Although the theorem focus on square symmetric matrices, the results can be extended to rectangular matrices. Let consider a rectangular matrix B then one can construct a symmetric square block matrix A from B:

$$A = \begin{pmatrix} 0 & B \\ B & 0 \end{pmatrix}$$

and apply results on A. When recovering A, the algorithm will also produce an approximation of B with the same error.

A.4 Other Approaches

Minimizing the trace norm in order to recover the input matrix proved to be a very efficient algorithm in number of needed samples since it converges with $O(nr\log^2 n)$ samples. However these solutions have trouble to scale with very large matrices that occur frequently in collaborative filtering. Later works have

Algorithm 12 OptSpace

Input: dataset $\mathcal{D}$, $M^{\mathcal{D}} \doteq M$ filled with 0 on unknown entries
Trim $M^{\mathcal{D}}$ and let $\bar{M}^{\mathcal{D}}$ be the ouput
Compute SVD of $\bar{M}^{\mathcal{D}} = U_0 S V_0^T$
Clean residuals errors by minimizing the discrepancy $F(U, V)$ initialized in U_0, V_0 (see equation A.5)

then try to reach the minimal number of samples, while others focus on alleviating the dependence on the condition number that could be problematic in practice. Finally, some works were also dedicated to matrix completion with side information.

Matrix Completion from a Few Entries
In 2009, in an algorithm called *OptSpace*, [Keshavan et al., 2009], [Hulikal Keshavan, 2012], proposed to recover the input matrix by first computing an SVD decomposition on a matrix whose rows and columns that contained more information than the average rows or columns where entirely set to 0. Although surprising, this technique makes the underlying low rank structure of the matrix more apparent. The intuition is that over-observed rows or columns can lead to overestimation of the top singular values and thus lead to poor initial points. This is somehow related to the overfitting concept. Then, they refined the decomposition by minimizing a function (by gradient descent) defined over the cartesian product of two Grassmann manifolds which minimizes the square distance between the reconstruction and the known entries (see algorithm 12:

$$F(U, V) = \min_{S \in \mathbb{R}^{K \times K}} \mathcal{F}(U, S, V) \tag{A.5}$$

$$\mathcal{F}(U, S, V) = \frac{1}{1} \min_{i,j \in \mathcal{D}} (M_{ij} - (USV^T)_{ij})^2 \tag{A.6}$$

Under incoherence condition and provided that $\forall (i, j) \in [n] \times [p], |\sum_{k=1}^{K} U_{ik}(\sigma_k/\sigma_1)V_{jk}| \leq \mu K^{1/2}$, they showed that if $O(nK\kappa_M^2 \log n)$ entries were revealed uniformly at random then their algorithm converges to the true matrix M with high probability. Apart from the condition number we see that algorithm 12 obtains the lower bound on the number of entries to be revealed at random of $O(nK \log n)$. However this algorithm can be quite computationally heavy and only guarantees asymptotic convergence unlike alternating least square solutions.

Matrix Completion from Noisy Entries Later, the authors extend the analysis of the *OptSpace* algorithm (with a slight modification with a regularization in F) to the case of approximately low rank matrix that $\tilde{M} = M + N$ where M is low-rank and N is small in some sense ([Keshavan et al., 2010]). They show that with $O(n\kappa_M^2 \max(K \log n, \kappa_M^4 K^2))$ entries (which again disregarding the condition number dependence achieved the $O(nK \log n)$ lower bound) one can obtain the following reconstruction error bound:

$$\frac{1}{\sqrt{np}}||\hat{M} - M||_F \leq C\kappa_M^2 \frac{p\sqrt{pK}}{n|\mathcal{D}|}||N^{\mathcal{D}}||_2 \tag{A.7}$$

Fast Exact Matrix Completion with Finite Samples

Several works have tried to obtain close to optimal results using the well used alternating least square algorithm. However results showed dependence on the condition number of the matrix that can be arbitrarily large. Different approaches have been investigated to improve these performances. For instance [Jain and Netrapalli, 2014] relied on using Singular Value Projection on rank-k matrix in a loop over k (see Algorithm 2 of [Jain and Netrapalli, 2014] for the detailed algorithm). Their main result is as follow:

Theorem 18 (Theorem 1 of [Jain and Netrapalli, 2014]). *Let $M \in \mathbb{R}^{n\times p}$ be a rank-K, μ-incoherent matrix (definition 14). Also, let each entry of M be observed uniformly and independently with probability q, if*

$$q > C\alpha \frac{\mu^4 K^5 (n+p)\log^3(n+p)}{p}$$

where $\alpha > 1$ and $C > 0$ is a global constant. Then with probability $1 - n^{-10-\log\alpha}$, the output $\hat{M}$ of their Algorithm 2 satisfies $||\hat{M} - M||_F \leq \varepsilon$. Moreover the run time of the algorithm is $O(|\mathcal{D}|r^2 \log(1\varepsilon)$

Using Side Information

Several work studied the problem of matrix completion assuming the existence of side information. For instance one could have gender, age, or location information about each users and price, color, category, or provenance information for the items. With good side information they proved that the number of samples needed to reconstruct the matrix can be drastically reduce. For instance [Xu et al., 2013] proved that only $O(\log n)$ information are necessary to recover the matrix but under strong assumptions. They assumed the existence of two matrices $A \in \mathbb{R}^{n\times r_a}$ and $B \in \mathbb{R}^{n\times r_b}$ that respectively contains users and items side information, with $\min(r_a, r_b) \geq r$. They further assumed that the true matrix M and the matrices A and B share the same latent information, that is the span of the column vectors of A (resp. B) contains the span of rows (resp. columns) of M. Those assumptions are really strong and hard to meet in practice. Thus their objective is to find the small matrix $Z \in \mathbb{R}^{r_a\times r_b}$ satisfying:

$$\begin{aligned} &\text{minimize} && \text{trace}(Z) \\ &\text{such that} && \forall (i,j) \in \mathcal{D}, [AZB^T]_{ij} = M_{ij} \end{aligned} \tag{A.8}$$

[Chiang et al., 2015] relaxed those assumptions by assuming that the side information may not be perfect and thus look for a solution that balance between classic matrix completion with and without side information:

$$\min_{Z \in \mathbb{R}^{r_a \times r_b}, X \in \mathbb{R}^{n \times p}} \sum_{i,j\mathcal{D}} \ell([AZB^T]_{ij} + X_{ij}, M_{ij}) + \lambda_Z ||Z||_* + \lambda_X ||X||_* \tag{A.9}$$

where ℓ is any loss function (most of the time the square loss) and λ_Z, λ_X parameters to balance between side information or not. $\lambda_Z = \infty$ will force Z to be 0 and thus side information will have no impact, and on the contrary $\lambda_X = \infty$ will put all the learning on the side information. Let $\mathcal{X}$ the constant such that $||X||_* \leq \mathcal{X}$, then with $O(\min(\mathcal{X}\sqrt{n}, \mathcal{X}^2 \log n)/\varepsilon^2)$ entries one can recover an ε approximation of M. This bound shows condition on the goodness of the side information. Indeed, with very good side information, X will be close to 0 since all the learning will be done by Z and thus $\mathcal{X}$ will be really low. However with really bad side information, all the learning will be done by X and $||X||_* \sim ||M||_* = O(n)$ so one would need $O(n^{(3/2)})$ entries to recover the matrix which is consistent with past results.

	Algorithm	Sample Complexity	Computation Complexity
[Candès and Recht, 2008] (2008)	NN	$O(n^{1.2} K \log n)$	$O(n^3)$ (per iter)
[Candès and Tao, 2009] (2009)	NN	$O(nK \log^6 n)$	$O(n^3)$ (per iter)
[Recht, 2009] (2009)	NN	$O(nK \log^2 n)$	$O(n^3)$ (per iter)
[Jain et al., 2013] (2012) ⋆	ALS	$O(\kappa_M^4 K^{2.5} n \log n \log \frac{K\|\|M\|\|_F}{\varepsilon})$	$O((\|\mathcal{D}\|K + nK^3) \log(1/\varepsilon))$
[Gunasekar et al., 2013] (2013)	ALS	$O(\kappa_M^6 K^7 n \log n \log \frac{\|\|M\|\|_F}{\varepsilon})$	$O((\|\mathcal{D}\|K + nK^3) log(1/\varepsilon))$
[Hardt, 2013] (2014) ⋆	ALS	$O(nK(K + \log(n/\varepsilon)\gamma_K^5 \left(\frac{\|\|M\|\|_F + \|\|V\|\|_F/\varepsilon}{\sigma_K}\right)^2)$	$O((\|\mathcal{D}\|K + nK^3) log(1/\varepsilon))$
[Keshavan et al., 2010] (2009)	OptSpace	$O(\kappa_M^2 nK \log n)$	$O(\|\mathcal{D}\|K \log n)$ (per iter)
[Keshavan et al., 2010] (2012) ⋆	OptSpace	$O(\kappa_M^2 nK \log n)$	$O(\|\mathcal{D}\|K \log n)$ (per iter)
[Jain and Netrapalli, 2014] (2014)	SVP	$O(nK^5 \log^3 n)$	$O(nK^7 \log^3 n \log(1/\varepsilon))$

Table A.1: Comparisons of various results on matrix completion on sample complexity and computation complexity. ε is the error between the recovered matrix and the true matrix in the Frobenius norm. $\gamma_K = 1 - \frac{\sigma_{K+1}}{\sigma_K}$. ⋆ indicates that noise was taken into account. NN = Nuclear Norm.

A.5 Conclusion

We have seen different works that studied the matrix completion problem from a theoretical point of view. Nuclear norm minimization approaches succeeded to prove that only $O(nK \log n)$ entries were needed to recover the entire matrix. This bound was closely approached by alternating least square algorithms but never met. However all those approaches suffer from several issues when applied to collaborative filtering. First the incoherence assumptions on the matrix can be a too strong hypothesis for collaborative filtering since it implies that all users (resp. all items) are really close, which limits the interest of performing individual recommendation. Second, papers on alternating least square algorithm did not take regularizations into account and were sometimes left as future work. Third, in collaborative filtering entries are not reveal at random, some users are much more active than others and likewise some items are much more popular than others. Thus some part of the matrix can be better reconstructed than other parts. Some researches take another direction and focused on the problem of sampling actively relevant entries in the matrix ([Bhojanapalli et al., 2014, Wang and Singh, 2015, Chakraborty et al., 2013]) instead of sampling randomly, and removed incoherence condition. Indeed incoherence condition implies that every entry amount for the same level of information leaving no space for active sampling. We believe that a very good understanding of matrix completion and active sampling can lead to very interesting algorithms in order to improve cold-start performances.